SOC TEST DATA VOLUME MINIMIZATION WITH TESTING TIME CONSTRAINT

Thesis submitted in the partial fulfillment of requirement for the award of degree of

Master of Technology

in

VLSI Design & CAD

Submitted By:

Deepika Sharma

Roll No.: 601061008

Under the Guidance of

Ms. Harpreet Vohra

Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

(Established under the section 3 of UGC Act, 1956)

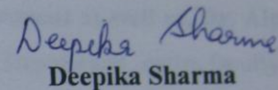
PATIALA – 147004 (PUNJAB)

June-2012

CERTIFICATE

I hereby declare that the work which is being presented here in the thesis entitled, " **SOC TEST DATA VOLUME MINIMIZATION WITH TESTING TIME CONSTRAINT** " in partial fulfilment of the requirement for the award of degree of M.Tech (VLSI Design and CAD) at Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Ms. Harpreet Vohra, Assistant Professor, ECED.

The matter presented in this thesis has not been submitted in any other University/Institute for the award of my degree.

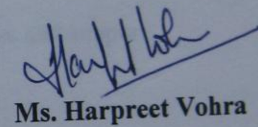


Date: 26/06/12

Deepika Sharma

Roll No.: 601061008

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

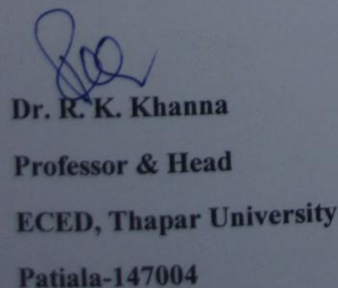


Ms. Harpreet Vohra

Assistant Professor

ECED, Thapar University

Counter Signed By:



Dr. R. K. Khanna

Professor & Head

ECED, Thapar University

Patiala-147004

Dr. S. K. Mohapatra

Dean Academic Affairs

Thapar University

Patiala-147004

ACKNOWLEDGEMENT

I would like to express my gratitude to all those who have helped me in this project. Firstly I owe my gratitude to my guide **Ms. Harpreet Vohra, Assistant Professor**, Electronics and Communication Engineering Department, Thapar University, Patiala whose help, stimulating suggestions and encouragement helped me throughout the time of research and writing of this report.

I am also thankful to **Dr. R. K. Khanna, Head of the Department** as well as, **Dr. Kulbir Singh, PG Coordinator,** Electronics and Communication Department, entire faculty and staff of Electronics and communication engineering Department and my friends who devoted their vauable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work.

Lastly, I would like to thank my parents and my brother for their love and support.

Deepika Sharma

601061008

ABSTRACT

Imperfections in the manufacturing process of very large scale integrated (VLSI) circuits give rise to manufacturing defects that result in improper circuit operation and which must be screened by the application of test patterns. Scan-based testing reduces the complexity of sequential automatic test pattern generation (ATPG) to combinational ATPG by providing full controllability/observability to the internal state elements. While providing clear benefits in terms of test preparation and fault coverage through structural test of complex system-on-a-chip (SOC) designs, the huge volume of test data (VTD) will render scan based testing ineffective due to limitations of the automatic test equipment (ATE). Although the ATE memory buffers can be reloaded, this will be achieved at the expense of increased time the SOC spends on the ATE, thus raising the cost of test. To overcome the VTD problem, test data compression is employed.

The reduction in test data volume will not only reduce ATE memory requirements, but also lower testing time. The testing time of an SOC depends on the test data volume, the time required to transfer the data to the cores, the rate at which the test data is transferred (measured by the cores test-data bandwidth and ATE channel capacity), and the maximum scan chain length. The total test time can be reduced by either reducing the test data volume or by shortening and reorganizing the scan chains. While test data volume reduction techniques can be applied to both hard and soft cores, scan chains cannot be modified in hard cores. Lower testing time will increase production capacity as well as reduce test cost and time to market for SOCs. Therefore new techniques are needed for decreasing test data volume in order to overcome memory bottlenecks and to reduce testing time.

Test-data compression offers a promising solution to the problem of reducing the test data volume for SOCs, especially if the cores are not BIST ready. Different compression techniques have been proposed over the years to reduce the test data volume. In this Thesis the multi code compression scheme based on different run length compression techniques (e.g. Golomb, Frequency-directed run-length (FDR), Alternating run-length (AR) and Extended Frequency-directed run-length (EFDR)) and Huffman is used . To achieve higher

compression ratio, an attempt has been made to combine the best properties of all the above techniques so as to utilize all the properties in a single multi code compression scheme. Decoder design for Multi Code Compression Scheme has also been implemented using (tools) which can work for different encoding schemes.

TABLE OF CONTENTS

CERTIFICATE	(i)
ACKNOWLEDGEMENT	(ii)
ABSTRACT	(iii)
TABLE OF CONTENTS	(iv)
LIST OF FIGURES	(viii)
LIST OF TABLES	(xii)
ABBREVIATIONS	(xiv)
Chapter 1 – Introduction	1
1.1 Introduction	2
1.2 Chip Testing	3
1.2.1 Test Cost	4
1.2.2 Reducing Test cost	5
Chapter 2 – Literature Review	7
2.1 Characteristics of Test Data	7
2.1.1 Hamming Distance	8
2.1.2 Bit Stuffing	9
2.1.3 Difference vector	9

2.2 Input Test Data Compression 9

2.2.1 Run Length Coding 10

2.2.2 Statistical Coding (Selective Huffman) 10

2.2.3 Mixed RL Huffman Coding 14

2.2.4 Golomb Coding 15

2.2.5 FDR Coding 17

2.2.6 EFDR Coding 19

2.2.7 AR Coding 20

2.2.8 Cyclical Scan Register decompression Architecture 23

2.3 Compaction of Test Response 25

2.3.1 Using MISR as Signature Analyzer 25

2.3.2 X-Compaction 26

Chapter 3- Design and Implementation of Test Architecture 29

3.1 Overall Test Architecture 29

3.1.1 Double Hamming Distance based Reorder Scheme 30

3.1.2 Compression Algorithm 30

3.1.3 Compression Analysis 31

3.1.4 Analysis of Multi-code Compression 32

3.1.5 Multi-code compression 36

3.2 Test Data Compression 37

3.2.1 Finite State Machine Representation of Decompressor 42

3.3 X-compaction for output response compaction 45

3.4 Tools Used 45

Chapter 4- Results and Analysis 46

4.1 Results for Test compression 47

4.1.1 Circuit S298 47

4.1.2 Circuit S400 51

4.1.3 Circuit S1494 55

4.1.4 Circuit S1196 59

4.1.5 Comparison of results for various compression schemes 61

4.2 Design Vision Results for Decompressor 65

4.3 Design Vision Results for Output side Compaction 74

Chapter 5- Conclusion and Future work 76

REFERENCES 77

LIST OF FIGURES

Figure 1.1 Scan Chain 2

Figure 1.2 Download of test patterns from Automatic Test Equipment 3

Figure 1.3 Test data compression Design for Testability 5

Figure 2.1 Testing of a module and its test patterns 8

Figure 2.2 Activity of pins of test sets 8

Figure 2.3 Example of Test Set Divided into 4-bit Blocks 11

Figure 2.4 Huffman tree for the code shown in Table 2.1 12

Figure 2.5 Huffman Tree for the 3 Highest Frequency Symbols in Table 2.1 12

Figure 2.6 Block Diagram of Huffman decoder 13

Figure 2.7 Block Diagram of Multi-bit Huffman decoder 14

Figure 2.8 Block Diagram of Golomb decoder 17

Figure 2.9 Block Diagram of FDR decoder 19

Figure 2.10 Block Diagram of EFDR decoder 21

Figure 2.11 Comparison of FDR and AR coding 22

Figure 2.12 Block Diagram of AR decoder 23

Figure 2.13 Architecture of CSR decompressor 24

Figure 2.14 The decompression architecture using boundary scan registers 25

Figure 2.15 The decompression architecture for using internal scan chain and user defined scan element to replace CSR 25

Figure 2.16 Compactor circuit 27

Figure 2.17 Design and Test Flow with X-compactor 28

Figure 3.1 Overall Test Architecture Design 30

Figure 3.2 Double Hamming distance based reordering scheme 30

Figure 3.3 Comparison of code length based upon RunLength 31

Figure 3.4 An example for weight of one test vector 35

Figure 3.5 Architectural diagram of decompressor 38

Figure 3.6 State Diagram for 2-bit parallel Huffman decoder 42

Figure 3.7 State Diagram for bit serial Huffman decoder 43

Figure 3.8 State Diagram of intergrated decompressor of RunLength based decoder 44

Figure 4.1 Summary of ISCAS'89 S298 circuit using RunLength based code 47

Figure 4.2 Summary of ISCAS'89 S298 circuit using RunLength and proposed RunLength plus Huffman based code 48

Figure 4.3 Summary of ISCAS'89 S298 circuit using RunLength + Huffman (m=4) 49

Figure 4.4 Summary of ISCAS'89 S298 circuit using RunLength + Huffman (m=5) 50

Figure 4.5 Summary of ISCAS'89 S400 circuit using RunLength based code 51

Figure 4.6 Summary of ISCAS'89 S400 circuit using RunLength and proposed RunLength plus Huffman based code 52

Figure 4.7 Summary of ISCAS'89 S400 circuit using RunLength + Huffman (m=4) 53

Figure 4.8 Summary of ISCAS'89 S400 circuit using RunLength + Huffman (m=5) 54

Figure 4.9 Summary of ISCAS'89 S1494 circuit using RunLength based code 55

Figure 4.10 Summary of ISCAS’89 S1494 circuit using RunLength and proposed RunLength plus Huffman based code 56

Figure 4.11 Summary of ISCAS’89 S1494 circuit using RunLength + Huffman (m=4) 57

Figure 4.12 Summary of ISCAS’89 S1494 circuit using RunLength + Huffman (m=5) 58

Figure 4.13 Summary of ISCAS’89 S1196 circuit using RunLength based code 59

Figure 4.14 Summary of ISCAS’89 S1196 circuit using RunLength and proposed RunLength plus Huffman based code 60

Figure 4.15 Summary of ISCAS’89 S1196 circuit using RunLength + Huffman (m=4) 61

Figure 4.16 Summary of ISCAS’89 S1196 circuit using RunLength + Huffman (m=5) 62

Figure 4.17 Comparison of compression ratios of Golomb and Golomb plus Huffman coding scheme for ISCAS’89 benchmark circuits 63

Figure 4.18 Comparison of compression ratios of Golomb and Golomb plus Huffman coding scheme for ISCAS’89 benchmark circuits 63

Figure 4.19 Comparison of compression ratios of Golomb and Golomb plus Huffman coding scheme for ISCAS’89 benchmark circuits 64

Figure 4.20 Comparison of compression ratios of Golomb and Golomb plus Huffman coding scheme for ISCAS’89 benchmark circuits 64

Figure 4.20 RTL of bit serial FSM based Huffman decoder for ISCAS’89 S400 circuit 65

Figure 4.21 Bit serial FSM based Huffman decoder for ISCAS’89 S400 circuit 66

Figure 4.22 Block Diagram of serial Huffman decoder 67

Figure 4.23 RTL of 2-bit parallel FSM based Huffman decoder for ISCAS'89 s400 circuit 67

Figure 4.24 2-bit parallel FSM based Huffman decoder for ISCAS'89 s400 circuit synthesized 68

Figure 4.25 RTL of MCC decoder 69

Figure 4.26 MCC decoder after synthesis 70

Figure 4.27 Block Diagram of FSM based MCC Decoder 71

Figure 4.27 Block Diagram of FSM based RunLength + Huffman Decoder 71

Figure 4.28 RTL of RunLength + Huffman Decoder 72

Figure 4.29 RunLength + Huffman Decoder after synthesis 73

Figure 4.30 Synthesized X-compactor with 12 scan outs compacted to 6 scan outs 74

Figure 4.31 Synthesized X-compactor with 20 scan outs compacted to 10 scan outs 75

LIST OF TABLES

Table 2.1: Statistical Coding Based on Symbol Frequencies for Test Set 11

Table 2.2: Example of Golomb coding for m=4 12

Table 2.3: Example of FDR Coding 18

Table 2.4: Example of EFDR Coding 20

Table 2.5: Example of AR Coding 22

Table 3.1: Comparison of length of codewords in each run length encoding 33

Table 3.2: Comparison of weight on each run-length code 34

Table 3.3: Multi Code Compression Scheme 37

Table 3.4: The Truth Table of Control Signals 39

Table 3.5: The Truth Table of Control Signals for 4:9 Decoder 40

Table 4.1: Results for ISCAS’89 S298 circuit using RunLength based compression code for TETRAMAX ATPG patterns 47

Table 4.2: Results for ISCAS’89 S298 circuit using RunLength and proposed RunLength + Huffman compression code for TETRAMAX ATPG patterns 48

Table 4.3: Results for ISCAS’89 S298 circuit using RunLength + Huffman (m=4) for TETRAMAX ATPG patterns 49

Table 4.4: Results for ISCAS 89 S298 circuit using RunLength + Huffman (m=5) for TETRAMAX ATPG patterns 50

Table 4.5: Results for ISCAS’89 S400 circuit using RunLength based compression code for TETRAMAX ATPG patterns 51

Table 4.6: Results for ISCAS’89 S298 circuit using RunLength and proposed RunLength + Huffman compression code for TETRAMAX ATPG patterns 52

Table 4.7: Results for ISCAS’89 S400 circuit using RunLength + Huffman (m=4) for TETRAMAX ATPG patterns 53

Table 4.8: Results for ISCAS 89 S400 circuit using RunLength + Huffman (m=5) for TETRAMAX ATPG patterns 54
Table 4.9: Results for ISCAS'89 S1496 circuit using RunLength based compression code for TETRAMAX ATPG patterns 55
Table 4.10: Results for ISCAS'89 S1496 circuit using RunLength and proposed RunLength + Huffman compression code for TETRAMAX ATPG patterns 56
Table 4.11: Results for ISCAS'89 S1496 circuit using RunLength + Huffman (m=4) for TETRAMAX ATPG patterns 57
Table 4.12: Results for ISCAS 89 S1496 circuit using RunLength + Huffman (m=5) for TETRAMAX ATPG patterns 58
Table 4.13: Results for ISCAS'89 S1196 circuit using RunLength based compression code for TETRAMAX ATPG patterns 59
Table 4.14: Results for ISCAS'89 S1196 circuit using RunLength and proposed RunLength + Huffman compression code for TETRAMAX ATPG patterns 60
Table 4.15: Results for ISCAS'89 SS1196circuit using RunLength + Huffman (m=4) for TETRAMAX ATPG patterns 61
Table 4.16: Results for ISCAS 89 S1196circuit using RunLength + Huffman (m=5) for TETRAMAX ATPG patterns 62

ABBREVIATIONS

IC	Integrated Circuit
BIST	Built-in Self Test
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generator
SOC	System on Chip
RL	Run Length coding
CUT	Circuit Under Test
DUT	Device Under Test
LUT	Look Up Table
SSF	Single Stuck at Fault
CMOS	Complimentary Metal Oxide Semiconductor
VLSI	Very Large Scale Integration
ASIC	Application Specific Integrated Circuit
DFT	Design for Testability
RL	RunLength code
RL+4H	RunLength + Huffman (m=4)
RL+5H	RunLength + Huffman (m=5)

CHAPTER 1- Introduction

1.1 System-on-Chip

The technology today makes it possible for designers to increase the complexity of their designs such that an entire electronic system can be implemented on a single chip. This design paradigm is called System-on-Chip, often abbreviated SOC. To shorten the time-to-market and to design complex systems, SOC designers use previously designed modules. Such modules are often called (embedded) cores. Some examples of cores are FPGA1s, DSP2s, RAM modules, micro-controllers and microprocessors. The described design methodology is called core-based design and is common practice today.

1.2 Testing

Testing is done to ensure that a physical device manufactured from a synthesized design has no manufacturing defect. Testing is essential because the imperfect semiconductor manufacturing process introduces defects in devices. Defects are physical imperfections. They are the root cause of failure, i.e. incorrect circuit behavior. Testing is done to identify chips with a failure and to diagnose the reason for a failure in order to improve the manufacturing process. Flaws are imperfections not serious enough to cause failure during test, but serious enough to cause early life failure. Testing is also done to identify the weak chips, i.e. the chips with flaws. Stimulus test patterns can be generated manually by using a description of the correct functional behavior of the digital chip. However, manual test pattern writing is very expensive. Therefore, automation is required. An Automated Test Pattern Generation (ATPG) tool automatically generates test patterns by using (1) a model of the gates and the interconnection of the gates (e.g. a Verilog netlist), and (2) a model of the potential faults. A fault model models the logic faulty behavior due to a defect. For example, the Single Stuck-at Fault (SSF) model assumes that defect(s) will cause a single node to either be stuck-at-1 or stuckat-0. If the values in all flip-flops in a sequential design chip can be controlled to any specific value and if they can be observed easily, then test pattern generation is reduced to testing combinational logic. For this purpose, additional Design-for-Testability circuitry is added to the design. The Level-Sensitive Scan Design (LSSD) technique can be used to control and observe all flip-flops [2][4]. Figure 1.1 shows a

technique using concatenated positive edge-triggered flip-flops. The concatenated flip-flops are also called a scan chain. The scan enable signal determines the operating mode of the scan chain. If the scan enable signal is set to one, each flip-flop input is connected to the flip-flop output of the previous flip-flop in the scan chain. Thus, a flip-flop latches the value of the previous flip-flop at each positive clock edge (also called shifting). The ATE shifts the stimulus pattern into the scan chain by using the scan input that is latched by the first flip-flop in the scan chain. Next, the responses to the stimulus pattern are latched in the flip-flops at the next positive clock edge by setting the scan enable signal to 0. Subsequently, the ATE shifts the response pattern out of the scan chain through the scan output and concurrently shifts the next stimulus pattern into the scan chain through the scan input by setting the scan enable signal to 1.

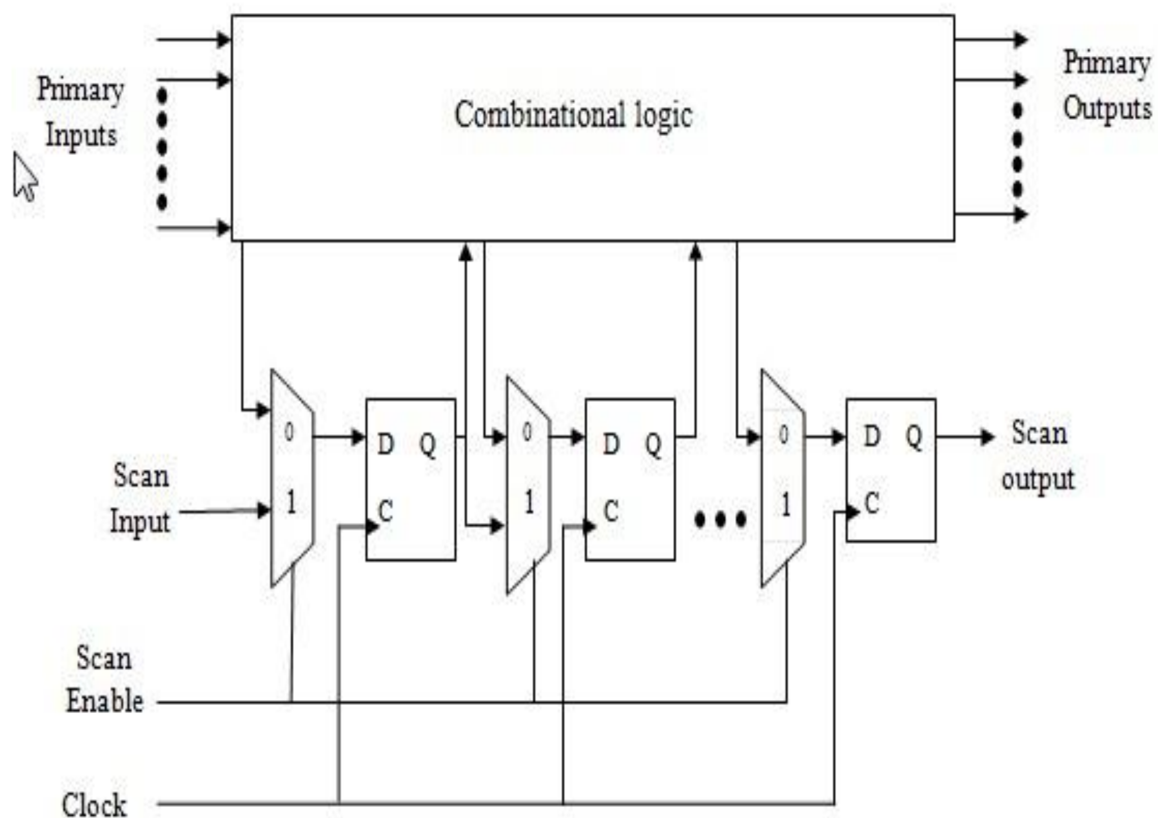


Figure 1.1: Scan Chain [2]

1.2 Motivation

With the advent of CMOS process, core-based design methodology is widely used to design a system-on-chip (SOC). However, SOCs present many test challenges, such as huge test data, test power, etc[1]. Due to the growth of complexity in an SOC, the volume of test data increases quickly**.** Test data is usually generated and stored on workstations. The increased variety of ASICs and decreased production volume of individual types of ASICs requires more frequent downloads of test data sets from workstations to automatic test equipment (ATE). In addition, because of the sheer size of test sets for ASICs, often as large as several gigabytes, the time spent to download test data from computers to ATEs is significant. The download from a workstation storing a test set to the user interface workstation attached to an ATE is often accomplished through a network. The download takes from several tens of minutes to hours. The test set is then transferred from the user interface workstation of an ATE to the main pattern memory through a dedicated high speed bus. The latter transfer usually takes several minutes. The transfer of test data from a workstation to an ATE is shown in Figure 1.2. During the download period of a test set, the ATE is idle, wasting this valuable resource. The overall throughput of an ATE is affected by the download time of test data, and the throughput becomes more sensitive to the download time with the increased variety of ASICs. To improve the throughput of an ATE, it is essential to reduce the download time of test data. A cost effective approach to achieve this is to compress the test data.

Automated Test Equipment (ATE*)* tests chips by (1) applying stimulus patterns to the chip inputs, (2) measuring the chip outputs, also called the response patterns, and (3) comparing the measured outputs to the correct outputs, also called the fault-free response patterns. An important reason for the increasing test cost is the increasing amount of test data (stimulus and response data). A promising solution is to store and transmit compressed stimulus data from the ATE to the chip. Additional circuitry on the chip decompresses the stimulus data and compresses the response data. Additional circuitry to make testing easier is called Design-for-Testability (DFT) circuitry.

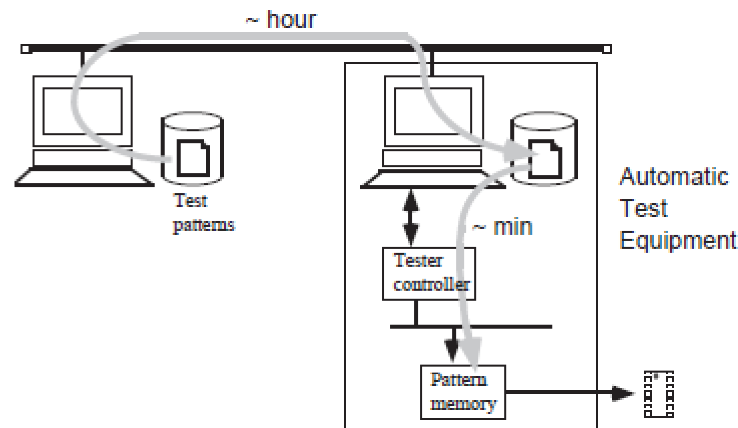


Figure 1.2: Download of test patterns to automatic test equipment [1]

1.3.1 Test Cost

Even though automated test pattern generation with scan chains reduces the cost of testing, the ever-increasing number of gates results in an ever-increasing number of test patterns. The increasing number of test patterns results in the following problems:

1. **Limited Bandwidth between Workstation and ATE**: The generated test patterns need to be uploaded from the workstation to the ATE memory. Due to the limited data bandwidth between workstation and ATE, this may take several tens of minutes to hours. During this time, the ATE remains idle. The cost of test increases with increasing idle time [3].
2. **Limited ATE memory**: An ATE with the required memory may not be available or may be very expensive. Therefore, test data is sometimes truncated, resulting in a reduced product quality [3].
3. **Limited Bandwidth between ATE and DUT**: The test patterns stored in the ATE memory need to be applied to the DUT (and the responses need to be observed by the ATE). Due to the limited data bandwidth between ATE and DUT, this may take a significant amount of (test) time. The cost of test increases with increasing test time [3].

1.3.2 Reducing Test Cost

Using Built-in-Self-Test (BIST) circuitry can reduce the test cost. The BIST circuitry generates all stimulus data, applies the data to the circuit, evaluates all response data, and determines whether or not a faulty response was produced. The premise of BIST is that no ATE is required. Therefore, BIST can also be used to test the product in the field. However, the area overhead of BIST can be significant (e.g., 3.4% [4]). Also the BIST design may depend on the test pattern set. Therefore, last minute design changes (resulting in test pattern modifications) may require re-designing the BIST hardware. This thesis focuses on compressing automatically (ATPG) generated test data. Test data compression enables transfer of compressed stimulus data from the ATE to the chip. Test compression circuitry decompresses the stimulus data and compresses the response data, see Figure 1.3.

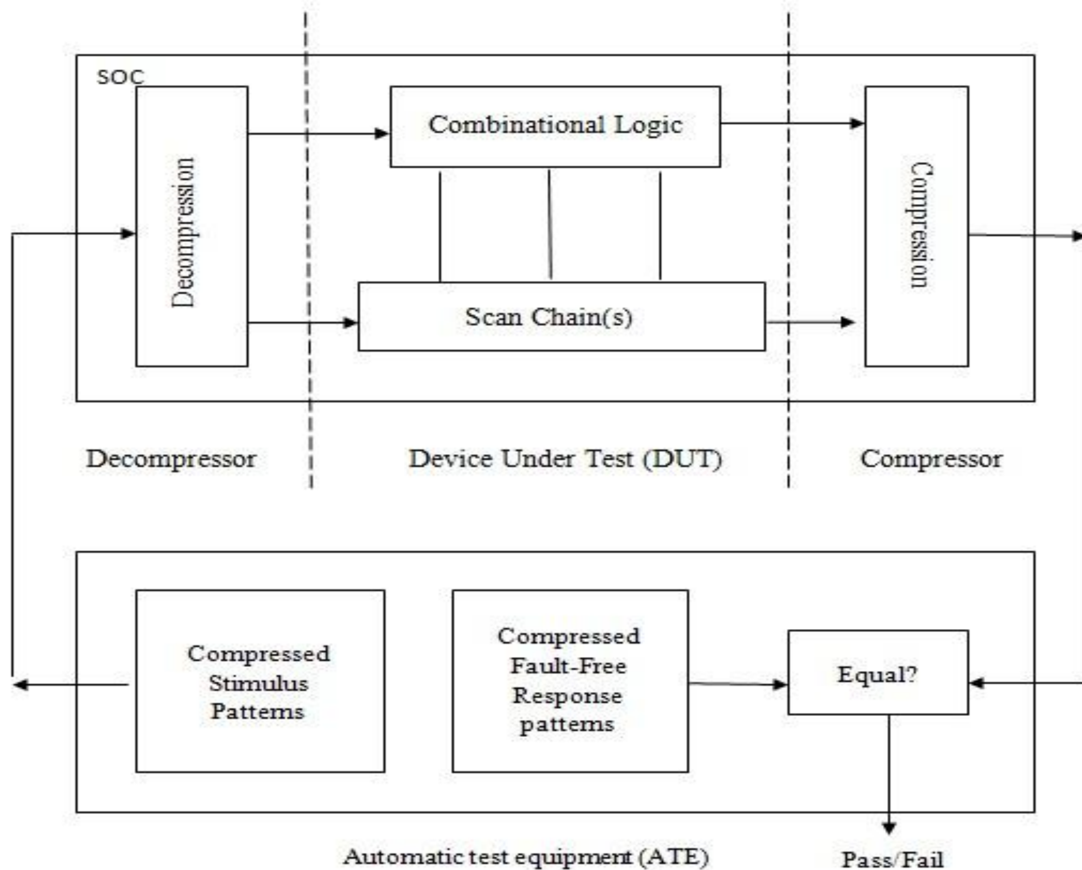


Figure 1.3: Test data Compression design for testability [2]

Test-data compression offers a promising solution to the problem of reducing the test-data volume for SoCs. In this approach, a pre-computed test set for an IP core is compressed (encoded) to a much smaller test set, which is stored in ATE memory. An on-chip decoder is used for pattern decompression to obtain from during test application.

CHAPTER 2 – Literature Review

Test data volume and its minimization for testing of a system has manifested itself as a serious problem. The testing time of a system majorly depends on test data volume which is a function of number of test patterns for that unit (input data) and the number of stimulus and response bits per pattern (output data). So in order to minimize test data, input data needs to be compressed and output data needs to be compacted. This chapter describes various schemes which had been proposed earlier for the test data minimization. Firstly the characteristics of test data will be described and then the methods based on those characteristics for the reduction of number of test patterns will be discussed and then the techniques for reduction of number of response bits per pattern will be focused.

2.1 Characteristics of Test Data

For a large complex circuit, a designer or a test generator generates test patterns considering one or a few modules at a time[1]. Therefore, a block of test patterns usually exercises a few modules of the circuit, while other modules are put under certain static conditions. This implies that logic values for only a subset of pins change for the block(s) of test patterns, while other pins are held at constant logic values. Figure 2.1 shows the testing of a module and a typical block of test patterns. In Figure 2.1, bold characters denote active pins whose logic value changes frequently and their logic values. All the other pins are held at constant values for the entire block of the test patterns except for the initial setup stage. Another useful observation is that the sequence of test patterns of an active pin often forms cycles. For example, Pin 4 in Figure 2.1 forms a cycle "X11X00," and the cycle repeats two times before it breaks. Pin 6 forms a cycle "X00," and it repeats three times. A test set is represented as a P×Q matrix, where P is the number of test patterns, and Q is the number of pins. A column number in the matrix corresponds to the pin number of the circuit. The activity of a column c_i is defined as the number of transitions on the i_{th} column of a test set, and is denoted as α (c_i). The activity of an example test set is given in Figure 2.2.

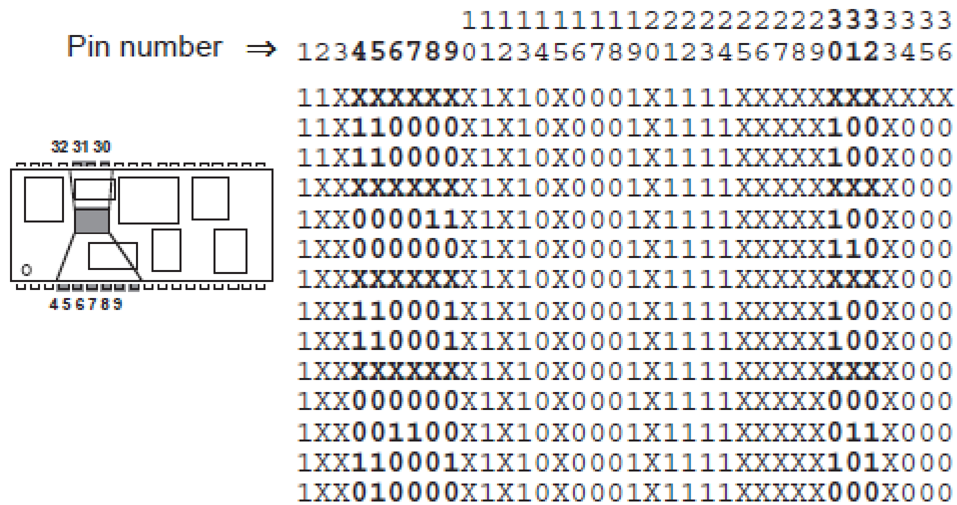


Figure 2.1: Testing of a module and its test patterns[1]

	1	2	3	4	5	6
	0	0	1	X	X	X
	0	0	X	1	X	1
	0	X	X	1	X	X
	0	X	1	1	X	1
	0	X	1	1	X	X
	0	X	0	X	X	0
$\alpha(c_i)$	**0**	**1**	**3**	**2**	**0**	**5**

Figure 2.2: Activity of pins of Test set [1]

Some of the features of test data which are useful for compression of test patterns are discussed below.

2.1.1 Hamming Distance

Given two bits i, j belong to{0, 1,X}, i and j are incompatible if i = 0 and j =1 or vice versa[18]. All other circumstances are compatible. The hamming distance is the distance between two scan frames is equal to the number of corresponding incompatible bits. For example, given two frames F1 = (10XX01) and F2 = (001X11), the distance d(F1, F2) is 2 because the first and the fifth corresponding bits in the frames are incompatible.

2.1.2 Bit Stuffing

Bit stuffing is the insertion of one or more bits into a transmission unit as a way to provide signaling information to a receiver [18]. The receiver knows how to detect and remove or disregard the stuffed bits. In VLSI testing bit stuffing is done in a test vector in such a way that it generates maximum number of zeroes to give maximum compression with various codes used for compression. For example, the vector is 10110X00XXX010. So here all don't care bits are replaced by zeroes and new bit stuffed vector will be 10110000000010.

2.1.3 Difference Vector

If T_D ={ t_1; t_2; . . . ; t_n} be the (ordered) pre-computed test set. T_{diff} is defined as follows: T_{diff} = {d_1;d_2; . . . ; d_n} = { t_1; t_1 XOR t_2; t_2 XOR t_3 ;……..; t_{n-1} XOR t_n }; where a bit-wise exclusive-or operation is carried out between patterns t_i and t_{i+1} . The successive test patterns in a test sequence often differ in only a small number of bits [18]. Therefore, T_{diff} contains few 1s and it can be efficiently compressed.

For given example:

The first vector is: 10110000000010

Second Vector: 11110000100010

The next vector will be difference of first and second vector.

Diff. Vector: 01000000100000

It is shown that the number of zeroes has increased and hence the run length of zeroes has also increased which in turn gives higher compression.

2.2 Input Test Data Compression

Input Compression also known as input compaction, relies on the possibility that, for a multiple output circuit, some of the outputs might be dependent on some of the inputs and not all. Thus, considering a full exhaustive test to be applied, the amount of test vectors required reduces to half per such independent input. As a hypothetical example, for a 7 input circuit, if three of inputs controlled only a subset of the outputs, the total amount of required exhaustive test patterns would be, in the worst case: $2^3 + 2^4 + 2^5 = 56 < 2^7 = 128$[5]. However, there is inherent assumption in this application, any possible interference between the disjoint

inputs are discarded, and therefore, the two sets cannot be said to have exactly same effectiveness.

Based on the characteristics of input data pattern explored and analyzed in the [1], the following compression schemes were formulated and applied to the input test patterns to achieve input test data volume minimization.

2.2.1 Run Length Coding

A sequence of identical symbols in a string is called a run. Run-length coding compresses data by representing each run into two tuples, the repeating symbol in the run and the run length [1]. A string "aabbbbcccd" has four runs, aa, bbbb, ccc, and d. It is coded as (a,2), (b,4), (c,3), (d,1). Run-length coding is simple in compression and in decompression. It is efficient for pins with low activity.

2.2.2 Statistical Coding (Selective Huffman)

In statistical coding, variable length code words are used to represent fixed-length blocks of bits in a data set. For example, if a data set is divided into 4-bit blocks, then there are 24 or 16 unique 4-bit blocks. Each of the 16 possible 4-bit blocks can be represented by a binary codeword. The size of each codeword is variable (it need not be 4 bits). The idea is to make the code words that occur most frequently have a smaller number of bits, and those that occur least frequently to have a larger number of bits. This minimizes the average length of a codeword. The goal is to obtain a coded representation of the original data set that has the smallest number of bits. A Huffman code [11] is an optimal statistical code that is proven to provide the shortest average codeword length among all uniquely decodable variable length codes. A Huffman code is obtained by constructing a Huffman tree. The path from the root to each leaf gives the codeword for the binary string corresponding to the leaf. An example of constructing a Huffman code can be seen in Table 2.1 and Figs. 2.4 and 2.5. An example of a test set divided into 4-bit blocks is shown in Fig. 2.3. Table 2.1 shows the frequency of occurrence of each of the possible blocks (referred to as symbols). There are a total of 60 4-bit blocks in the example in Fig. 2.3. Figure 2.4 shows the Huffman tree for this frequency distribution and the corresponding code words are shown in Table 2.1.

0010 0100 0010 0110 0000 0010 1011 0100 0010 0100 0110 0010
0010 0100 0010 0110 0000 0110 0010 0100 0110 0010 0010 0000
0010 0110 0010 0010 0010 0100 0100 0110 0010 0010 1000 0101
0001 0100 0010 0111 0010 0010 0111 0111 0100 0100 1000 0101
1100 0100 0100 0111 0010 0010 0111 1101 0010 0100 1111 0011

Figure 2.3: Example of Test Set Divided into 4-Bit Blocks [11]

Sym.	Freq	Pat.	Huff. Code	Sel. Code
S_0	22	0010	10	10
S_1	13	0100	00	110
S_2	7	0110	110	111
S_3	5	0111	010	00111
S_4	3	0000	0110	00000
S_5	2	1000	0111	01000
S_6	2	0101	11100	00101
S_7	1	1011	111010	01011
S_8	1	1100	111011	01100
S_9	1	0001	111100	00001
S_{10}	1	1101	111101	01101
S_{11}	1	1111	111110	01111
S_{12}	1	0011	111111	00011
S_{13}	0	1110	-	-
S_{14}	0	1010	-	-
S_{15}	0	1001	-	-

Table 2.1: Statistical Coding Based on Symbol Frequencies for Test Set in Fig. 2.3 [11]

An important property of Huffman codes is that they are prefix-free. No codeword is a prefix of another codeword. This greatly simplifies the decoding process. The decoder can instantaneously recognize the end of a codeword uniquely without any look-ahead. The amount of compression that can be achieved with statistical coding depends on how skewed the frequency of occurrence is for the different code words. If all of the code words occur with equal frequency, then no compression can be achieved. It is well known, however, that the test vectors in a test set tend to have a lot of correlations. This arises from the fact that faults in the CUT that are structurally related require similar input value assignments in order

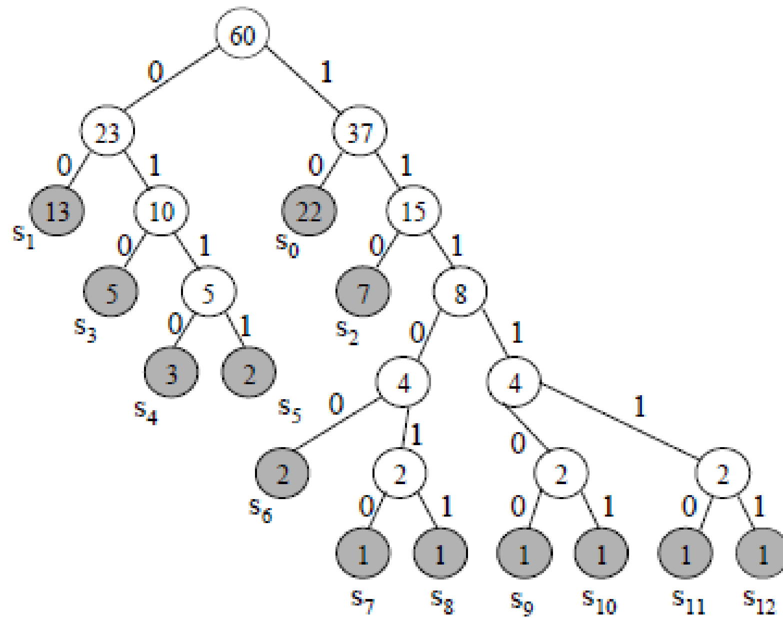


Figure 2.4: Huffman Tree for the Code Shown in Table 2.1 [11]

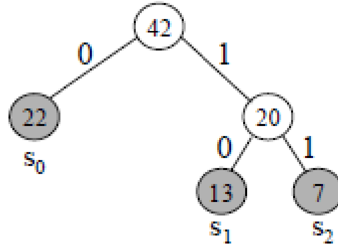


Figure 2.5: Huffman Tree for the 3 Highest Frequency Symbols in Table 2.1 [11]

to be provoked and sensitized to an output. This often results in skewed frequency of occurrence for different code words. Moreover, for test cubes (which are test vectors that are not fully specified, i.e., contain X's), the compression can be very large. The X's provide flexibility to allow a block to be encoded with more than one possible codeword. The shortest possible codeword can be chosen for each block to maximize the compression. To fully exploit the correlations in a test set, the number of bits in each scan vector should be a multiple of the fixed-length block size used for the statistical code. When dividing the test set into *b*-bit blocks for coding, if the size of the scan vectors is not a multiple of *b*, then X's can be added to pad the start of the vectors (first bits shifted into the scan chain) to make the

length a multiple of *b*. Shifting some extra bits (at the start of the vector) into the scan chain doesn't matter provided the final contents of the scan chain contains the correct test vector when it is applied to the core-under-test. Having each scan vector be a multiple of the block size aligns the blocks within the vectors so that the correlations between the bit will skew the frequencies. The block diagram of a huffman decoder is given below

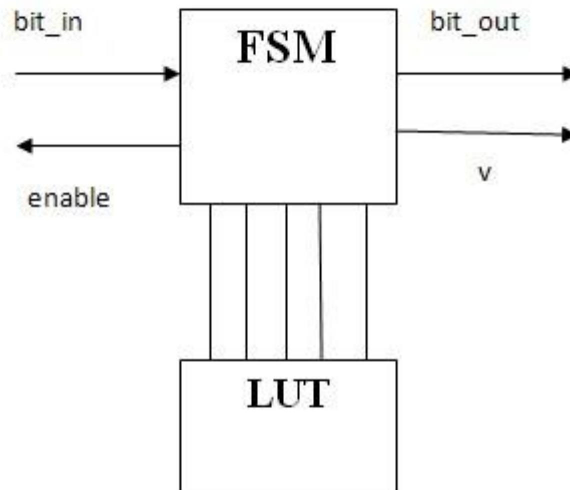


Figure 2.6: Block Diagram of Huffman Decoder [11]

However the throughput rate of this kind of decoder is very less.Given a conventional Huffman decoder tree structure, a single input bit will be decoded each cycle. Since the average length of each code word is H bits/symbol, on average, 1/H symbols would be decoded per clock cycle. Thus the expected throughput rate is 1/H symbols per clock. With the fixed K-bit look ahead Huffman Decoding, K bits of input stream will be examined during each clock cycle. Therefore, the throughput rate increases.

Optimal multibit look ahead huffman decoder maximizes the expected decoding throughput rate subject to resource constraints [22]. The block diagram of the multibit huffman decoder is shown in figure 2.7. It consists of a buffer, barrel shifter, FSM block and look ahead table. It can encode K bits per cycle. If one wants to decode K bits per cycle, then each state will have

2^K going arcs to other states even the total number of states remains unchanged. The corresponding state table will have $N.2^K$ entries, where N is the number of states and hardware implementation cost will grow exponentially with respect to K.

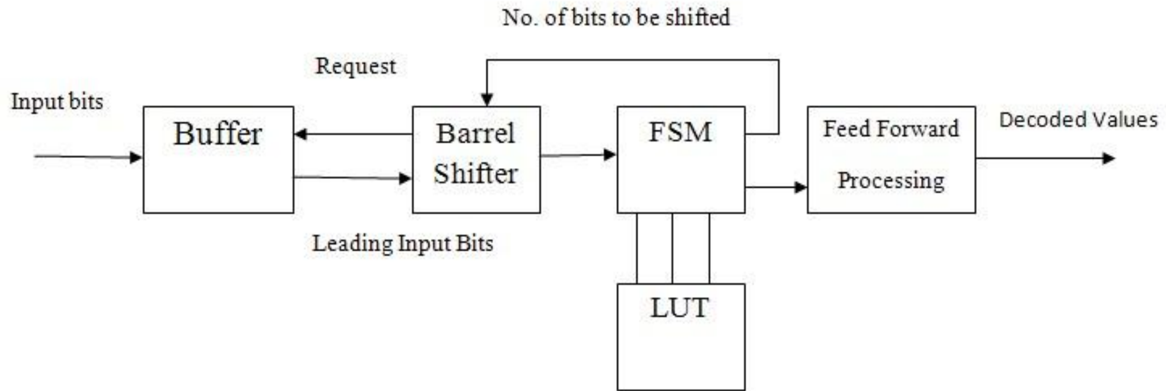


Figure 2.7: Block Diagram of Multi-bit Huffman Decoder

2.2.3 Mixed RL-Huffman Coding

This technique mixes two encoding techniques, called RL Huffman [12], to reduce test data volume, test application time and scan test power dissipation. This is achieved in two steps. The first step applies the Run-Length (RL) to the test vector set. Specifically, it minimizes the transitions and eventually power dissipation during test and also compresses data by grouping 0's and 1's in blocks. The second step produces more compression on top of the first step using Huffman encoding.

In the first step, the RL encoding is used for both power reduction and first round of data compression. In RL encoding, instead of sending the exact bits of a test vector, the length of the created blocks are sent. Hence, a test vector is partitioned into some variable length of 0-blocks or 1-blocks. Having don't care bits in the test vectors, there are different ways of creating 0-blocks and 1-blocks because of replacing the don't cares with either '0' or '1'. Therefore, replacing the don't care bits is an important issue which causes different results of coding such as more compression and power reduction.

The RL encoding not only reduces the test power dissipation but also performs the compression. To get the best compression rate Huffman coding (a variable length encoding by nature) [11] to encode block length values (characters). The idea is to assign smaller number of bits to the code words that occur most frequently and larger number of bits to those that occur less frequently. A Huffman code is obtained by constructing a Huffman tree.

2.2.4 Golomb Coding

The method [13] is especially suitable for encoding pre-computed test sets for embedded cores in a system-on-a-chip (SoC). The major advantages of Golomb coding of test data include very high compression, analytically predictable compression results, and a low-cost and scalable on-chip decoder. It allows multiple cores in an SoC to be tested concurrently using a single automatic test equipment input–output channel.

For any give sequence of difference vectors, tight upper and lower bounds are derived on the amount of compression that can be obtained with Golomb codes. Similar bounds for conventional run-length coding in order to highlight the inherent superiority of Golomb codes are also derived. The first step in encoding a test set is to generate its difference vector test set. The next step in the encoding procedure is to select the Golomb code parameter m, referred to as the group size. The choice of m has received a lot of attention in the information theory literature—for certain distributions of the input data stream (T_{diff} in our case), the group size m can be optimally determined. For example, if the input data stream is random with zero probability, then m should be chosen such that $p^m = 0.5$. However, since the difference vectors for pre-computed test sets do not satisfy the randomness assumption, the best value of m for test-data compression must be determined experimentally. Nevertheless, the best value of m can be approximated analytically. Once the group size is determined, the runs of zeros in T_{diff} are mapped to groups of size m (each group corresponding to a run length). The number of such groups is determined by the length of the longest run of zeros in T_{diff}. The set of run lengths {0,1,2…..m-1} forms group A_1. The set {m, m+1,….2m-1} form group A_2 etc. In general the set of run lengths {(k-1)m,(k-1)m+1,(k-1)m+2…….,km-1} comprises group A_k. To each group A_k a group prefix of (k-1) ones is assigned followed by a zero. If m is chosen to be a power of two, i.e. $m = 2^N$ each group

contains 2^N members and $\log_2$ m-bit sequence(tail) uniquely identifies each member within the group. Thus the final code for a run length L that belongs to group A_k is composed of two parts a prefix and a tail.

The block diagram of the golomb decoder is shown in figure 2.8. It consists of 9 states Finite state machine (FSM) and log2m bit counter. The bit in is primary input, the encoded test data is sent via bit-in to FSM when the signal en is high. The inc is a count-up signal to notice counter, as it finishes counting, the rs signal/line will show high condition. The out represents the output decoded data. When v is high which means the output data is valid. The movement of decoder is that the counter counts up to m cycles to enable the next input data is ready to

Group	Run-Length	Group Prefix	Tail	Codeword
A_1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A_2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A_3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011
….	…..	…..	…..	…….

Table 2.2: Example of Golomb coding for m=4 [13]

be sent. Moreover, the v is high as it outputs m 0‘s while the counter is doing count-up. On the other hand, when bit-in is 0, the counter stops, thus, FSM follows the tail, the part of codeword, and sends out few 0s and signal 1 but then the movement of decompression would be completed in orders.

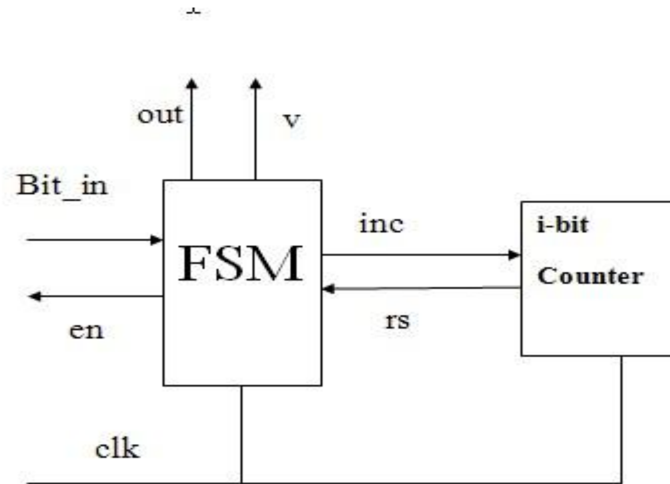


Figure 2.8: Block diagram of Golomb Decoder [13]

2.2.5 FDR Coding

An FDR code [14] can be used to compress both the difference vector sequence T_{diff} and the test set T_D. Let $T_D = \{t_1, t_2, t_3, \ldots, t_n\}$; be the (ordered) pre-computed test set. The ordering is determined using a heuristic procedure described later. T_{diff} is defined as follows:

$$T_{diff} = d_1, d_2, \ldots, d_n = \{t_1, t_1 xor\ t_2, \ldots\ldots, t_{n-1} xor\ t_n\}$$

where a bit-wise exclusive-or operation is carried out between patterns t_i and t_{i+1}. This assumes that the CSR starts in the all-0 state. (Other starting states can be considered similarly). The successive test patterns in a test sequence often differ in only a small number of bits. Therefore, T_{diff} contains few 1s and it can be efficiently compressed using the FDR code. However, the test architecture requires additional CSR and an exclusive-or gate for pattern decompression. If the uncompacted test set T_D is used for compression, all the don't-care bits in TD are mapped to 0s to obtain a fully specified test set before compression.

The FDR code is constructed as follows: The runs of 0s are divided into groups A_1; A_2;A_3; . . .;A_k, where k is determined by the length l_{max} of the longest run ($2^k - 3 < l_{max} < 2^{k+1} - 3$). Note also that a run of length l is mapped to group A_j where $J = \lceil log_2(l+3) - 1 \rceil$ i.e. The size of the i_{th} group is equal to 2^i, i.e., A_i contains 2^i members. Each codeword consists of two parts—a group prefix and a tail. The group prefix is used to identify the group to which the

run belongs and the tail is used to identify the members within the group. The encoding procedure is shown in Table 2.3.

The block diagram of FDR which consists of three parts: 9-state FSM, k-bit serial input counter, and log2 k-bit counter shown in Figure 2.7, The bit_ in is inputting encoded data when the en is high, and the out outputting decoded data is valid while the v is high. The counter-in is a path which sends data to k-bit counter, and the shift is a signal that takes control of the counter-in. Dec1 is used to notice k-bit counter starting count-down, and the rs1 is high as the counting finishes. The dec2 tells $\log_2$ k-bit counter start doing count-down, inc, on the other hand, tells it starts counting up. The rs2 will tell when it finishes counting. The operation of circuit is shown as Figure 2.9.

Group	Run Length	Group Prefix	Tail	Code Word
A_1	0	0	0	00
	1		1	01
A_2	2	10	00	1000
	3		01	1001
	4		10	1010
	5		11	1011
A_4	6	110	000	110000
	7		001	110001
	8		010	110010
	9		011	110011
	10		100	110100
	11		101	110101
	12		110	110110
	13		111	110111
………….	………..	………….	………….	………..

Table 2.3: Example of FDR Coding

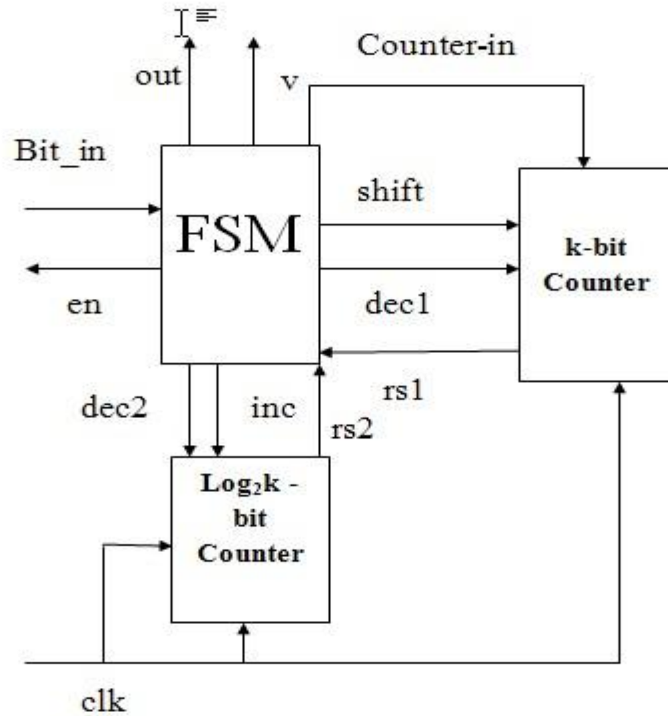


Figure 2.9: Block Diagram of FDR Decoder [14]

1. The FSM sends the data of prefix into the K-bit counter, signal en, shift and inc keep at high till it receipts the last bit, 0 to divide the prefix from tail.
2. After receiving the data from prefix, FSM enables dec1 and v high and outputs several 0s till the k-bit counter finishes counting down.
3. The $\log_2$ k-bit counter also counts the number of bit in prefix during the and the $\log_2$ k bit counter starts counting down the number of bit of tail which received by k-bit counter.
4. When the tail data in k-bit counter counts down, the out outputs a few of 0's and outputs a 1 after it finishes counting to achieve the decoding for a codeword.

2.2.6 Extended Frequency Directed Run-length (EFDR) Code

EFDR [15] method extends the function of encoding run 0s length in FDR to reach encoding 1's run length. The advantages are proposed in [15], Table 2.4 list the process of encoding which adds extra bits on the length of codeword. Moreover, the rest of codeword stay the

same. The extra bit must be located in MSB of codeword to inform the decoder whether it should invert the output data or not

Group	Run Length	Group Prefix	Tail	Code Word Runs of 0's	Code Word Runs of 1's
A_1	1	0	0	000	100
	2		1	001	101
A_2	3	10	00	01000	11000
	4		01	01001	11001
	5		10	01010	11010
	6		11	01011	11011
A_3	7	110	000	0110000	1110000
	8		001	0110001	1110001
	9		010	0110010	1110010
	10		011	0110011	1110011
	11		100	0110100	1110100
	12		101	0110101	1110101
	13		110	0110110	1110110
	14		111	0110111	1110111

Table 2.4: Example of EFDR Coding [15]

shown in Figure 2.10, before inputting prefix data, the extra bit will be input first to enable XOR gate by FSM via the mux signal, if the mux is 1, the out is inverse the fout data, if it's 0, the out is equal fout, through the signal mux can decode codeword which encode by run 0s run-length or run 1s run-length.

2.2.7 **Alternative Run-length (AR) code**

The AR coding also improves FDR coding to encode input test data by alternating 0's run length and 1's run lengths without adding length of codeword. Table 2.5 illustrates the

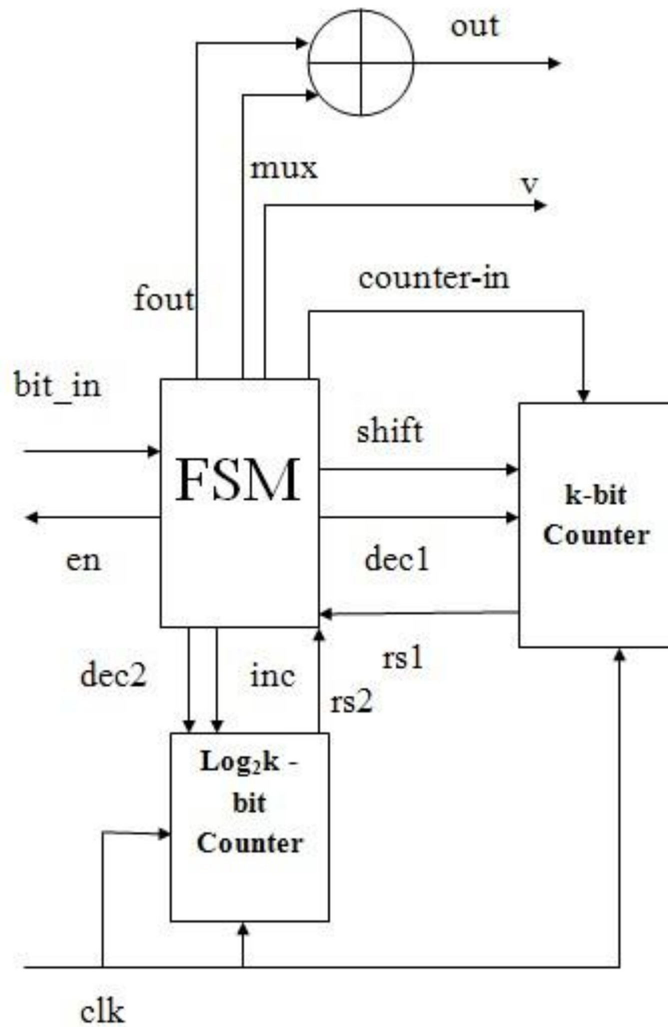


Figure 2.10: Block Diagram of EFDR Decoder [15]

process of AR coding, as we can see that the codeword of AR coding is the same as FDR coding whether the input test data is encoded by run-length of 0‘s or 1‘s. Table 2.5 is an example to explain which type can encode by run-length of 0‘s or 1‘s. Figure 2.11 is an example, there are 26 bits in input stream, nevertheless, the encoding result of FDR consists of 30 bits, over 4 bits, but the result of AR is 18 bits, thus AR coding is a efficient method in this case.

Group	Run Length of 1's	Run Length's of 0's	Group Prefix	Tail	Code Word
A1	1	1	0	0	00
	2	2		1	01
A2	3	3	10	00	1000
	4	4		01	1001
	5	5		10	1010
	6	6		11	1011
A3	7	7	110	000	110000
	8	8		001	110001
	9	9		010	110010
	10	10		011	110011
	11	11		100	110100
	12	12		101	110101
	13	13		110	110110
	14	14		111	110111

Table 2.5: Example of AR Coding [20]

Input data stream: 00000001111111111000000001 (26- bits)

FDR encoded data: 110001000000000000000000110010(30-bits)

AR encoded data: 110001110011110010 (18-bits)

| run 0 | | run 1 | | run 0 |

Figure 2.11: Comparison of FDR coding and AR coding [20]

When encoding run 0s length it always fetch segment which starts with 0, ends with 1. 1's run length, on the opposites, starts with 1, and ends with 0. On other hand, the detail description that the efficacy of power consumption and benefit of compression ratio present is discussed in [20].

The circuit diagram block of AR decoder is shown in Figure 2.12, there is additional T Flip-Flop with negative trigger and a XOR gate in FDR decoder. The operation of additional part is using signal feedback and negative trigger to control the point t, when t=1, the output data

of out is inverting fout, when t=0, the output data of out and fout is equal. The operation of other part is the same of FDR decoder.

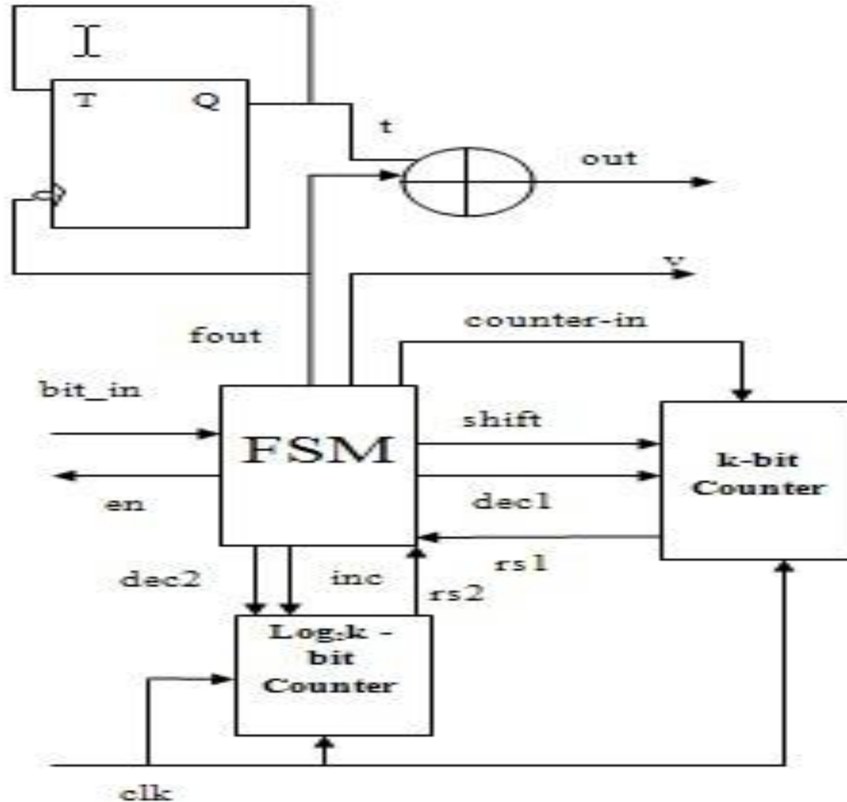


Figure 2.12: Block diagram of AR decoder [20]

2.2.8 Cyclical Scan Register Decompression Architecture

T_{diff} is generated by the T_D, if $T_D = \{ t_1, t_2, t_3, \ldots, t_n\}$ and $T_{diff} = \{ t_1, t_1 \text{ xor } t_2, t_2 \text{ xor } t_3, \ldots, t_{n-1} \text{ xor } t_n \}$, which will be compressed into T_E by using the Multi Code Compression Techniques . The decoding architecture is shown in Figure 2.13, the T_E will be decoded into T_{diff} by on chip decoder with run length coding, and using XOR gate and feedback from the cyclical scan register (CSR) [23] to decompress the T_{diff} into the T_D . The compression ratio of run length coding will be growing, because the process that transforms the T_D into the T_{diff} is increasing the length of run of 0's. Therefore, this encoding method and decoding

architecture are certainly increasing compression ratio of run length coding in most of the test set feedback from the cyclical scan register (CSR) to decompress the T_{diff} into the $T_{D.}$

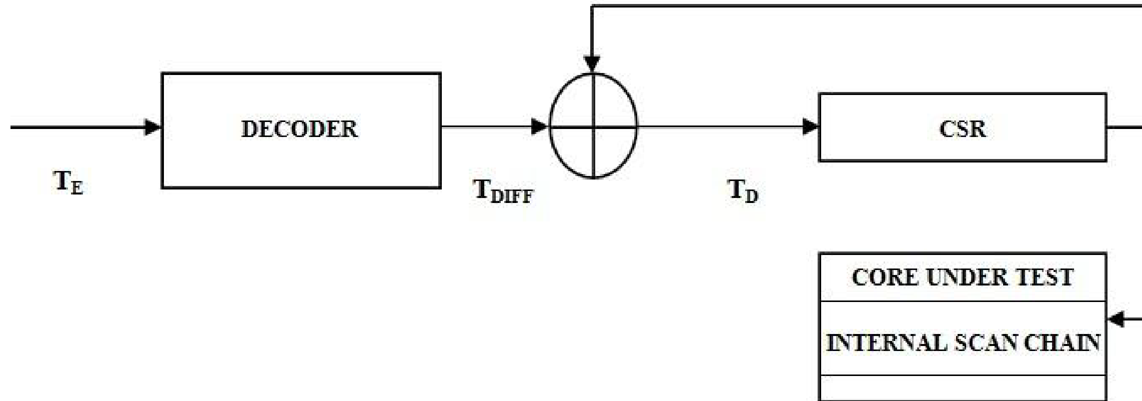


Figure 2.13: Architecture of CSR decompression [23]

The CSR decompression architecture can be changed by following the different circuit condition's in Figure 2.13, it is using boundary scan register to replace CSR, if the length of boundary scan register is longer than test pattern, then one can feedback the needed length of test pattern to XOR operation with the next test pattern. Thus, this technique can simultaneously save extra area of CSR to reach the purpose. Besides, the internal scan chain of the other cores also can be used if the core clock is different from the core under test and controlled. The architecture is given by Figure.2.14 used when the internal scan chain's length is equal to the test pattern or shorter than its length. If it is shorter, the user defined scan elements can be added to complete the test pattern length. The SCR decompression architecture can be changed by following the different circuit conditions, in Figure 2.13, it is using boundary scan register to replace CSR, if the length of boundary scan register is longer than test pattern, then we can feedback the needed length of test pattern to XOR operation with the next test pattern. Thus, this technique can simultaneously save extra area of CSR to reach the purpose. Besides, the internal scan chain of the other cores is can be used if the core clock is different from the core under test and controlled.

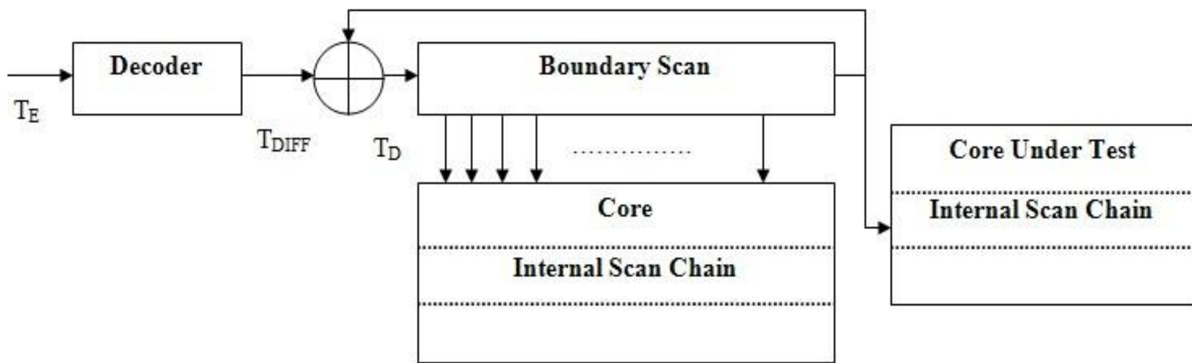


Figure 2.14: The Decompression architecture using Boundary Scan register [23]

The architecture is given by Figure 2.14 and used when the internal scan chains length is equal to the test pattern or shorter than its length. If it is shorter, the user defined scan elements can be added to complete the test pattern length.

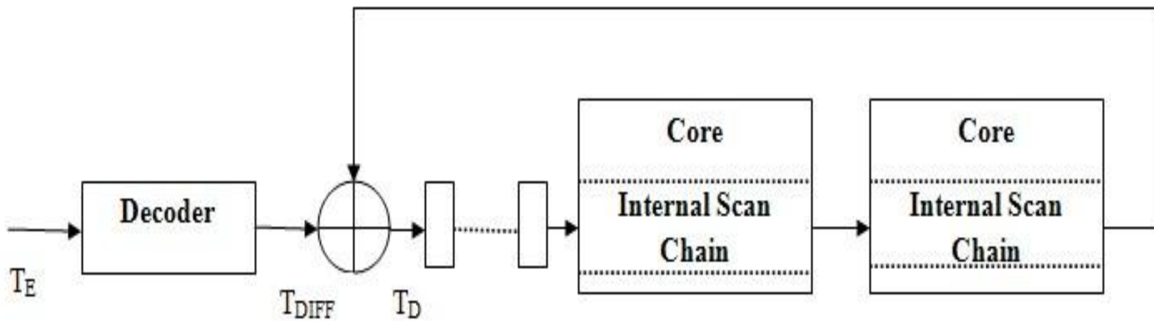


Figure 2.15: The decompression architecture for using internal scan chain and user defined scan element to replace CSR [23]

2.3 Compaction of test response

The conventional testing techniques of digital systems require application of test stimuli generated by a test pattern generator (TPG) to the circuit under test (CUT) and subsequent comparison of the produced responses with known correct responses. However, for large circuits, because of higher storage requirements for the fault-free responses, the procedure turns out to be rather expensive, and hence alternative approaches are sought. Test response data can be compressed by following two schemes:

2.3.1 Using MISR as Signature Analyzers

The scan chain outputs are connected to the MISR inputs and the MISR generates the signature of all test response bits [7]. Next the ATE compares the observed signature with the fault-free signature stored in the ATE memory and obtained by simulating all fault-free test responses. If and only if the signature is correct, the device passes the test. The problem arises when one of the test response bits are unknown (X). Unknown test response bits will result in multiple fault-free signatures. Therefore, the generated signature cannot be compared with one unique fault-free signature and the test may not be able to detect some erroneous test responses. Eventually, all possible signatures may become fault-free signatures, i.e., the test result will be invalid.

2.3.2 X-Compaction

X-Compact [7] is an X-tolerant test response compaction technique. It enables up to exponential reduction in the test response data volume and the number of pins required to collect test response from a chip. The compaction hardware requires negligible area, does not add any extra delay during normal operation, guarantees detection of defective chips even in the presence of unknown logic values (often referred to as X's), and preserves diagnosis capabilities for most practical scenarios. The technique has minimum impact on current design and test flows, and can be used to reduce test time, test-data volume, test-input/output pins and tester channels, and also to improve test quality. Using the X-Compact design technique, the number of scan-out pins can be reduced to $\log_2 n$ almost without requiring any information about the layout of the scan chains or the test vectors used. The X-compact technique does not compromise the error detection and diagnosis capability of scan-based DFT for all practical purposes even in the presence of unknown logic values (often referred to as X-values). The X-Compact design technique is independent of the fault models and test patterns used. The X-Compactor circuit has very little hardware overhead and does not affect system performance during normal operation. Insertion of X-Compactor circuits does not require any change to the automatic test pattern generation (ATPG) flow. The X-Compact technique can be mainly used for the following purposes: 1) to reduce test time by allowing a lot of scan chains with very few pins; 2) to reduce test data volume due to exponential reduction in the response data volume; 3) to reduce the number of input/output pins needed

for test purposes; 4) to improve the quality of test by making room for more test patterns; and 5) to test ICs on very low cost testers with very few scan channels. The figure 2.20 shows the X-Compactor design by which number of scan out pins can be significantly reduced thus reducing amount of test response to be stored.

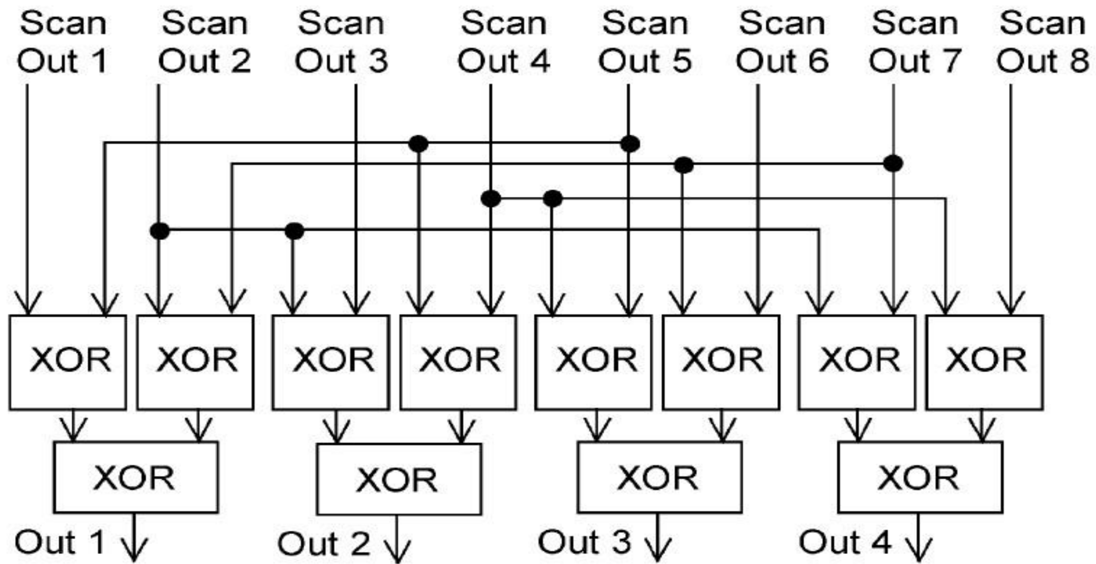


Figure 2.16: Compactor Circuit [7]

Test patterns must be applied to test the compactor circuit connected to the scan chain outputs. Test patterns can be applied to the compactor circuit inputs by scanning in the appropriate patterns using the scan chains and observing the compactor outputs on the tester. For generating patterns to test the compactor circuit, any conventional ATPG tool can be used. Since the compactor circuit consists of trees of XOR-gates, one needs only four test patterns such that each XOR gate in the compactor circuit is tested exhaustively. This is a well-known property of logic implemented as XOR-trees. While all single stuck-at faults are detected by these test patterns, some bridging faults, for example, may not be detected. However, it is very unlikely that a defect in the compactor circuit will not be detected by millions of vectors scanned out during the application of test patterns used to test the integrated circuit.

The insertion of compactor circuits has very little impact on the existing design and test flows. During design, scan chain insertion, and ATPG, we do not have to worry about the compactor circuits. After scan chain insertion is done and the design is "test ready," one has clear idea of the number of scan chains. Next, the technique is used to design an X-Compactor circuit and insert this circuit at the outputs of the scan chains. Finally, the compactor circuit needs to be simulated to obtain its responses to the scan chain outputs (i.e., the fault-free response computed by the ATPG tool for every scan-out cycle). This is because, when ATPG was performed, the compactor was not in the design. The simulation can be performed by using simple XOR operations or by using any commercial logic simulation tool. Figure 2.17 shows the overall flow.

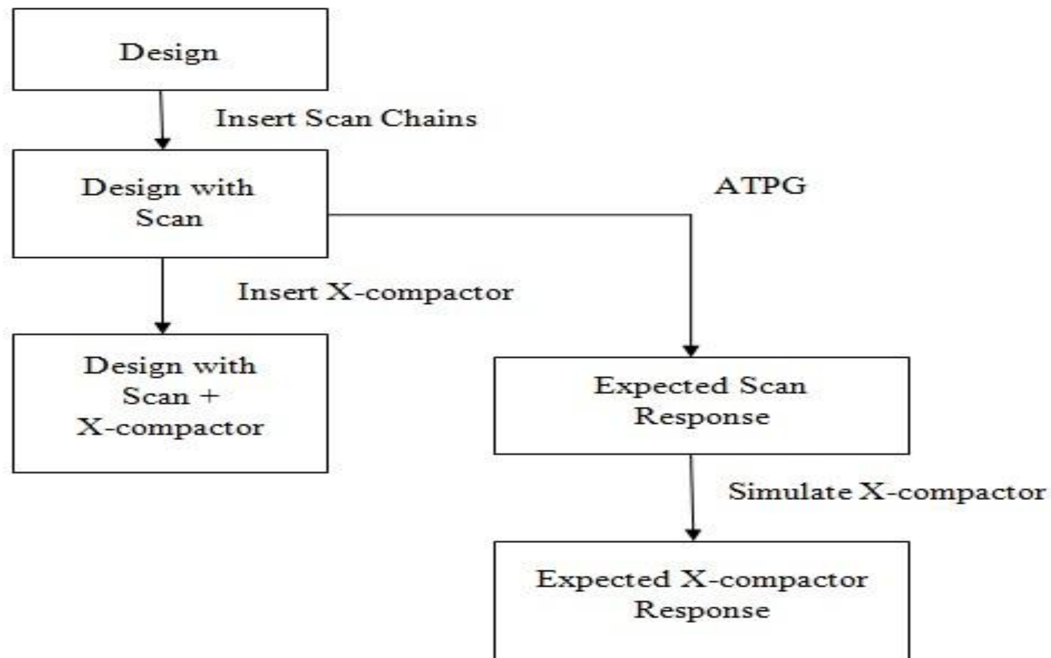


Figure 2.17: Design and test flow with X-compaction

It is clear from Figure 2.17 that the generation of a compactor design does not have to be dependent on the test patterns generated by ATPG.

CHAPTER 3 - Design and Implementation of Test Architecture

3.1 Overall Test Architecture

The Test architecture consists of automatic test equipment where compressed stimulus patterns are stored. Run-Length based and Huffman compression algorithms are used to compress the input test data to be stored. The SOC contains the decompressor which is used to decode the encoded data. FSM based decoder is used corresponding to the implementation details of compression algorithms used. At the output side, Output Compactor is used to compact response bits. X-compactor is used to realize output compaction and the data is stored in Output Response Analyzer.

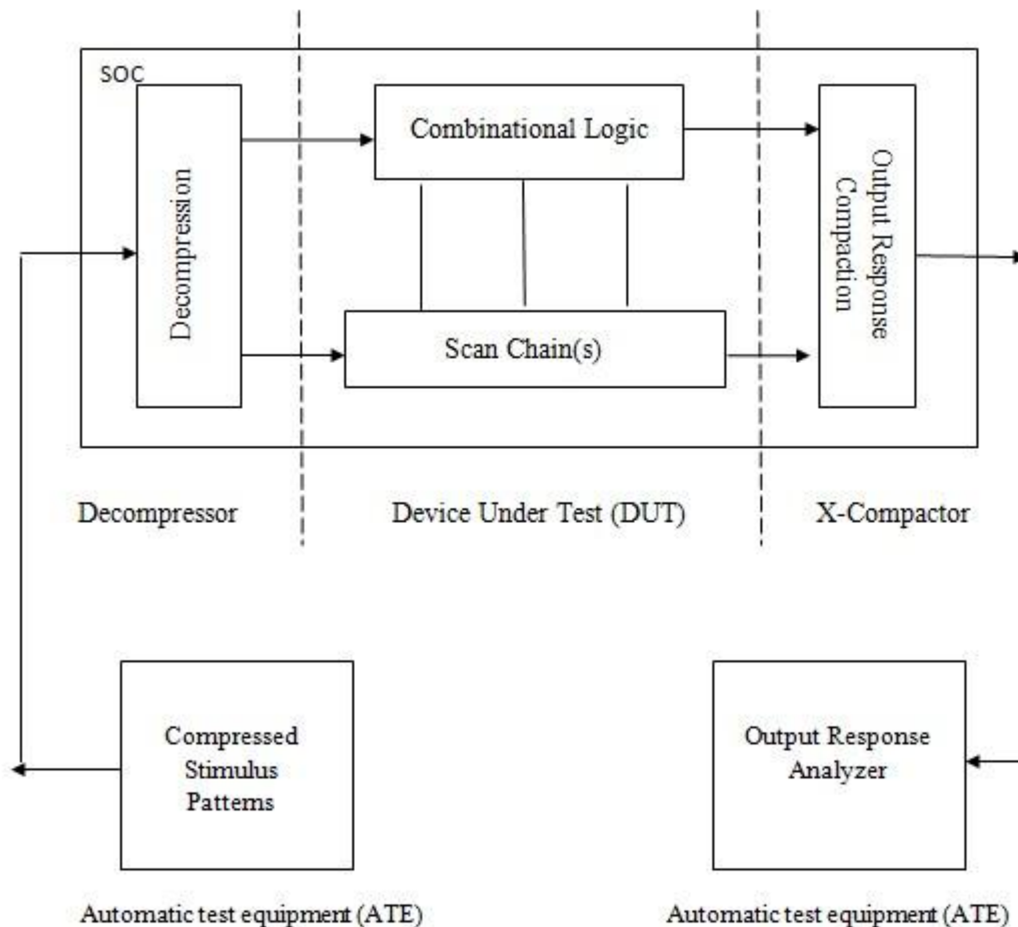


Figure 3.1: Overall Test Architecture Diagram

3.1.1 Double Hamming Distance Based Reorder Scheme

Hamming Distance Based Reordering Scheme is used [18] in order to have better compression ratio when it is used with multi code compression scheme. A flow chart depicting the scheme is shown in figure 3.2

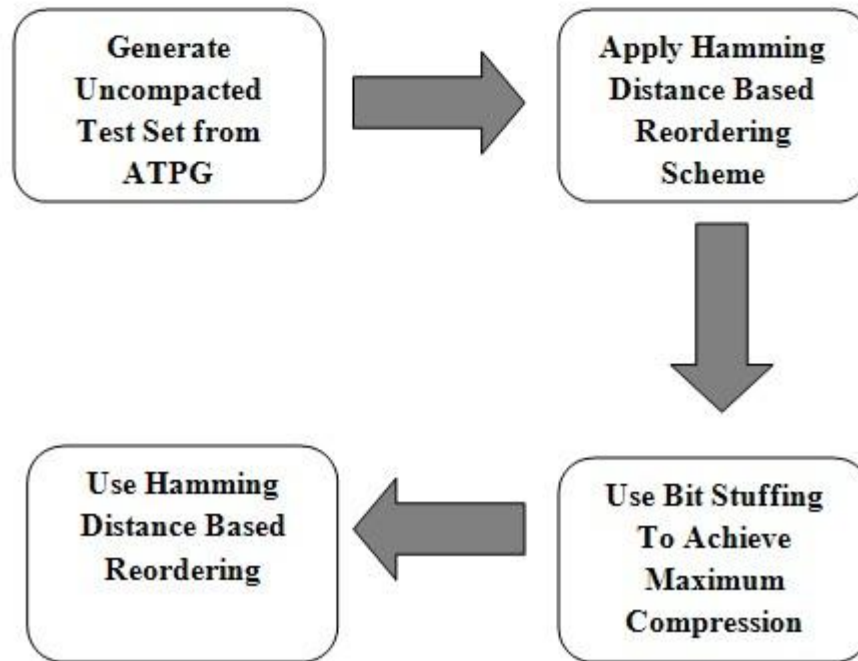


Figure 3.2: Double Hamming Distance Based Reordering Scheme

At first Test patterns are generates through ATPG. Then they are reordered with hamming distance reordering scheme and after that bit stuffing is used to achieve maximum compression ratio. Again hamming distance reordering is used in order to have greater lengths of zeroes.

3.1.2 Compression Algorithms

Two encoding techniques have been mixed to reduce test data volume and test pattern delivery time in scan test applications. This is achieved by using the Run-Length (RL) encoding [1] followed by Huffman encoding [12]. This combination is especially effective when the ratio of don't cares in a test set is high which is a common case in today's large SoCs. The first step applies the Run-Length (RL) to the test vector set. Specifically, it

minimizes the transitions and also compresses data by grouping 0's and 1's in blocks. The second step produces more compression on top of the first step using Huffman encoding. A Huffman code is obtained by constructing a Huffman tree. The blocks with higher occurrence frequencies will get the smaller length codeword. For this example, instead of sending 64 bits, ATE sends only 64-34=30 bits. There is 53.12% overall saving in test data volume and also transfer time.

3.1.3 Compression Analysis

Each test data compression method has the highest compression ratio for different circuits design with the same ATPG tool. Second, as we know, if the number of bits of codeword is fewer then the compression efficiency is excellent. Thus one can use lesser bits of codeword to represent longer original test data. For example consider the length of codeword in these four methods as shown in Figure 3.3. When the run length is ≤ 5, the FDR has the shortest codeword. On the other hand, when the number of run length is > 6, Golomb code (group size m = 16 or 32) has the shortest codeword. Moreover, EFDR and AR are extended from FDR, under each different status of 1's run lengths in test set. EFDR which adds on bits

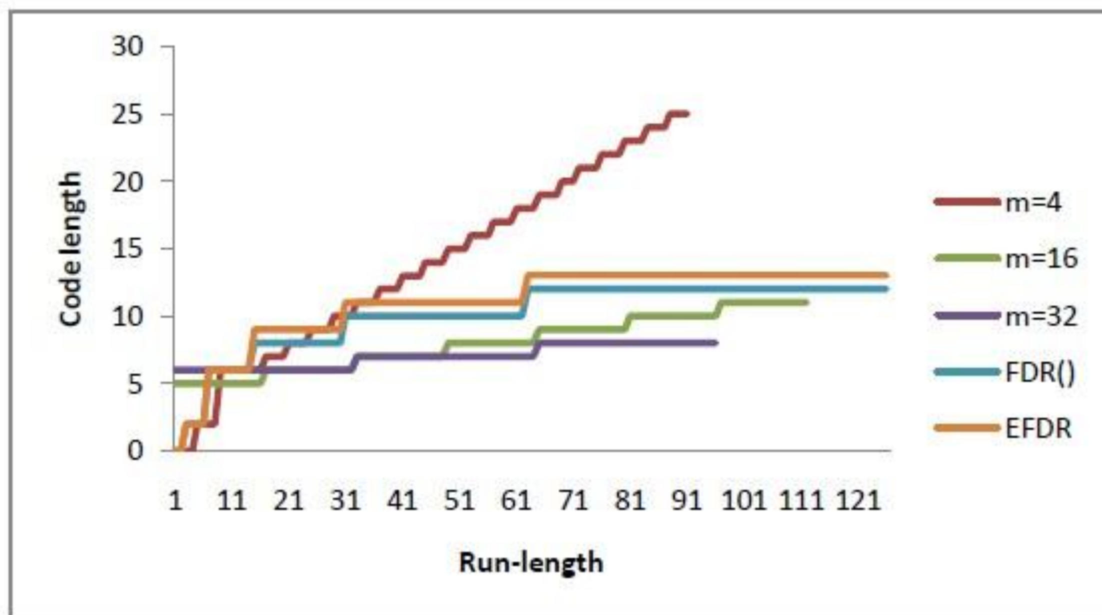


Figure 3.3: Comparison of code length based upon Run Length

length in codeword to stand for the encoding of 1's run length or 0's run length as well as AR, which interchanges 1's run length with 0's run length without any increase in length as its encoding method. According to the above comparison, suitable run-length based encoding method is used. After that the huffman compression algorithm is applied to achieve further compression. Thus, combined decoder is used which includes the decoding of run length based methods and Huffman scheme. Firstly, FSM based huffman decoder is used to decode the data and then MCC based decoder is used to decode the data obtained after huffman decoding process.

3.1.4 Analysis of Multi Code Compression

Detailed analysis of multi code compression method is described in this section. Let the pre-computed test set be $T_D = \{t_{d1}, t_{d2}, t_{d3} \dots t_{dn}\}$ which has equal numbers of bit as r from t_{d1} to t_{dn}, and let the compressed test data be $T_E = \{t_{e1}, t_{e2}, t_{e3} \dots t_{en}\}$. Compressed t_{e1} to t_{en} has different size due to run 1's or 0's distribution and the adequate run length encoding. Let us assume Golomb, FDR, EFDR and AR as G, F, E and A. The number of bit of t_{dn} as c, therefore, G_{c1} stands for the number bits in the 1st compressed test vector by Golomb. Next, with four kinds of run length method, the comparison on each number of bits in code word will be illustrated in Table 3.1 below. The first column gives the number of run length while the rest of them represent the number of bits. Each run length methods have its longer or shorter decoding status compared to the original data. Thus, it is assumed that in every test vector, k sub runs are contained. However, AR, due to the decoding method alternates between 0 or 1's, so the value of k is different from the other three methods. L here is used as the number of bit in every sub run, so $t_{d1} = \{L_1, L_2, L_3 \dots L_k\}$, $L_1 + L_2 + L_3 \dots + L_k = r$, and the first compressed test vector represented as $t_{e1} = \{u_1, u_2, u_3 \dots u_k\}$, u as the number of bit of every compressed sub runs. So that, $u_1 + u_2 + u_3 \dots + u_k = c_1$ But, the number of c_1 changes with different encoding methods. With every test vector as a unit, calculate the encoding efficiency by Weight (w). With the variety of run lengths, different codewords represent its corresponding method. We assume that in original, number of bits are os. csX represents the number of bit compressed by compression method X. The wX=os–csX. When the run length is y, wyX stands for the weight in every method, shown as in Table 3.2.

Obviously, the weight of each method in different run length can be easily told. For example, when run length is 4, Golomb (m =32) is 6 bit codeword. Os4 = 5, cs4G (m=32) = 6, we obtain w4G= -1 as result. Thus, the value of weight is w5F =2 when the FDR run length is 5. If the value of weight is positive, it shows this encoding method is effective. When it is 0 or negative, it shows the compression is ineffective. If the value of w is bigger, the encoding efficiency is good.

Original		**No. of bits of Codeword (cs)**				
Run Length	**No. of bits original (os)**	**Golomb m = 16 (csG)**	**Golomb m = 32 (csG)**	**FDR (csF)**	**EFDR (csE)**	**AFDR (csA)**
0	1	5	6	2	3	2
1	2	5	6	2	3	2
2	3	5	6	4	5	4
3	4	5	6	4	5	4
4	5	5	6	4	5	4
5	6	5	6	4	5	4
6	7	5	6	6	7	6
7	8	5	6	6	7	6
8	9	5	6	6	7	6
9	10	5	6	6	7	6
10	11	5	6	6	7	6
11	12	5	6	6	7	6
12	13	5	6	6	7	6
…	….	…..				

Table 3.1: Comparison of length of codewords in each run length encoding

Let a test vector be t_d, shown in Figure 3.4. When t_{d5} is encoded by Golomb, FDR and EFDR, this vector is separated into 9 sub runs, so k =9, on the other hand, when using AR as encoding method, k =17. However, the total number of test vectors weight can be obtained. In this case, FDR has positive value while others have negative ones. Therefore, FDR is the only method that compresses effectively. These results of four kinds of encoding methods are easily and effectively shown by calculating the weights. We can use every single test vectors weights to acquire the method that has higher compression ratios.

Original		**No. of bits of Codeword (cs)**				
Run Length	**No. of bits of original**	**Golomb m = 16**	**Golomb m = 32**	**FDR**	**EFDR**	**AFDR**
0	1	-4	-5	-1	-2	-1
1	2	-3	-4	0	-1	0
2	3	-2	-3	-1	-2	-1
3	4	-1	-2	0	-1	0
4	5	0	-1	1	0	1
5	6	1	0	2	1	2
6	7	2	1	1	0	1
7	8	3	2	2	1	2
8	9	4	3	3	2	3
9	10	5	4	4	3	4
10	11	6	5	5	4	5
11	12	7	6	6	5	6
12	13	8	7	7	6	7
….	**…..**	**……**				

Table 3.2: Comparison of weight on each run-length code

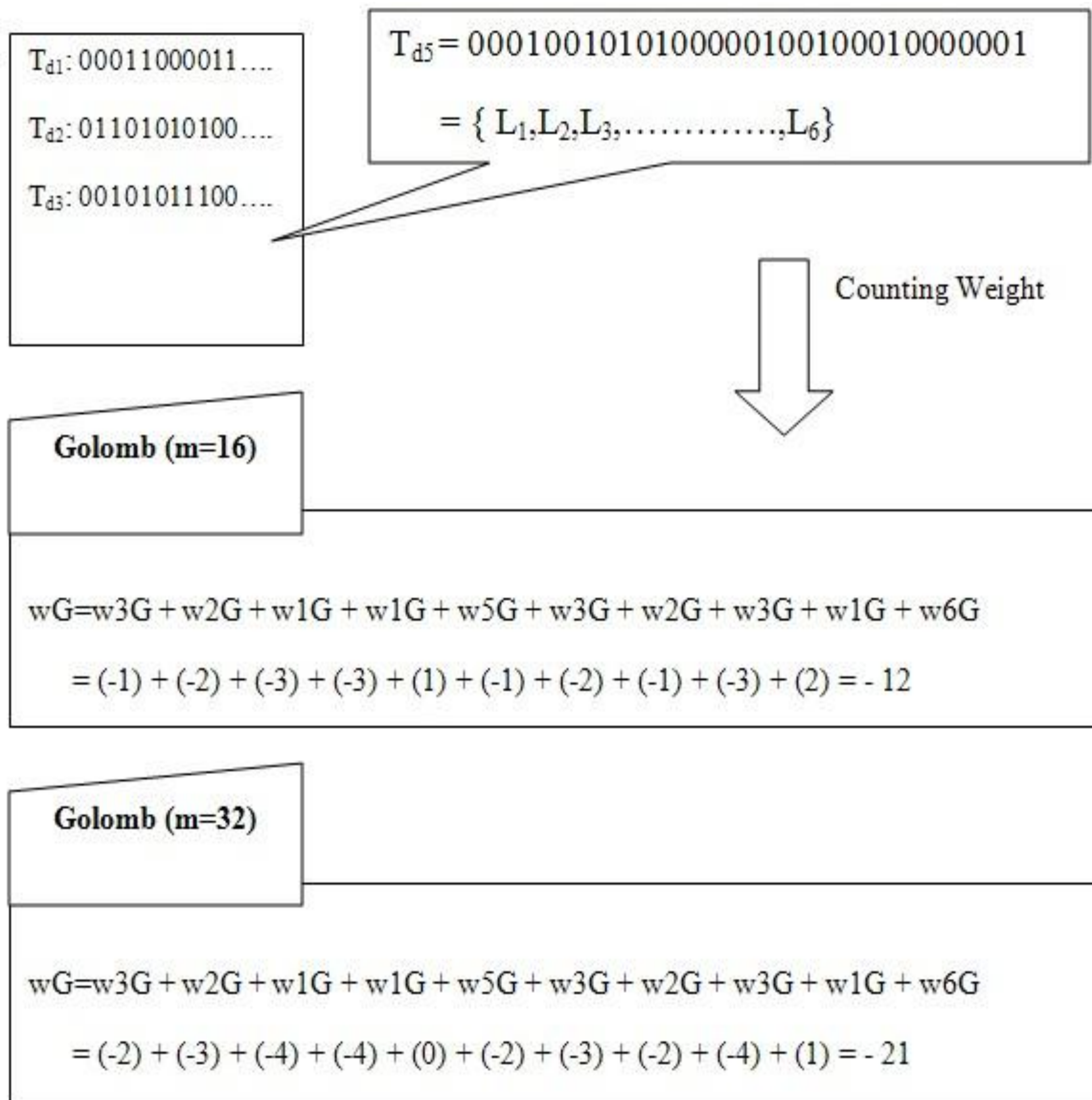


Figure 3.4: An example for weight of one test vector

3.1.5 Multi Code Compression

In the last section, we calculated each test vector‘s compression result in different encoding methods by weight. However, if we encode all the test pattern with this concept, it would soon have serious negative effect as extra hardware overhead rapidly increases but compression ratio does not proportionately becomes higher.

During compression, some conditions should be considered as well, such as the frequency and distribution of 1‘s and 0‘s run length in test set, the required hardware cost for decompression and etc. Therefore we generate difference vector test set (T_{diff}) with the test pattern reordering along with the double Hamming distance based Reorder scheme so as to increase compression in testing.

With a careful observation of all the test data compression variable to variable run length codes we have managed to conclude on certain properties of each encoding which are listed as under.

- For l=0 to 2 no compression technique provides a positive compression for both run length of either 0’s or 1’s.
- For l=3 to 5 FDR and IFDR provides highest compression for run length of 0’s and 1’s respectively.
- For l>5 GOLOMB provides highest compression for a run length of 0’s.
- For l>5 for sequence of 1’s we can compare AR, EFDR and IFDR for highest compression and utilize with maximum compression.

So it‘s better to leave small run length of 0‘s or 1‘s as it is without compressing to get an higher percentage of compression ,rather an positive compression in various cases and thus a state called BYPASS MODE is introduced in the compression scheme.

Run Length of 0's	Compression Scheme Used	Run Length of 1's	Compression Scheme Used
0	No Scheme Used Bypass Mode	0	No Scheme Used Bypass Mode
1	No Scheme Used Bypass Mode	1	No Scheme Used Bypass Mode
2	No Scheme Used Bypass Mode	2	No Scheme Used Bypass Mode
3	FDR	3	IFDR or FDR or AFDR
4	FDR	4	IFDR or FDR or AFDR
5	FDR	5	IFDR or FDR or AFDR
6 or Higher	GOLOMB	6 or Higher	IFDR or FDR or AFDR

Table 3.3: Multi Code Compression Scheme

3.2 Test Data Decompression

In this section decompressor architecture is introduced which is integrates huffman and four kinds of run length decoders. This Decoder not only conform to requirement of different cores via switching control signal, but it also decreases area overhead by hardware sharing. The decompressor decompresses the encode test set T_E and outputs difference vectors T_{diff}. The XOR gate and the CSR are used to generate the test pattern from the T_{diff}.

The architectural diagram of the decompressor is shown in figure 3.5. It is comprised of a bit serial huffman decoder, 4-16 decoder, a p-bit counter, a q-bit counter, a D_{out} block and a FSM block. Serial Huffman decoder's output serves as input to FSM block. The data is first decoded by the huffman decoder and the decoded data from this decoder goes to the FSM block so that it can be further decompreesed based on the run length bases code using the

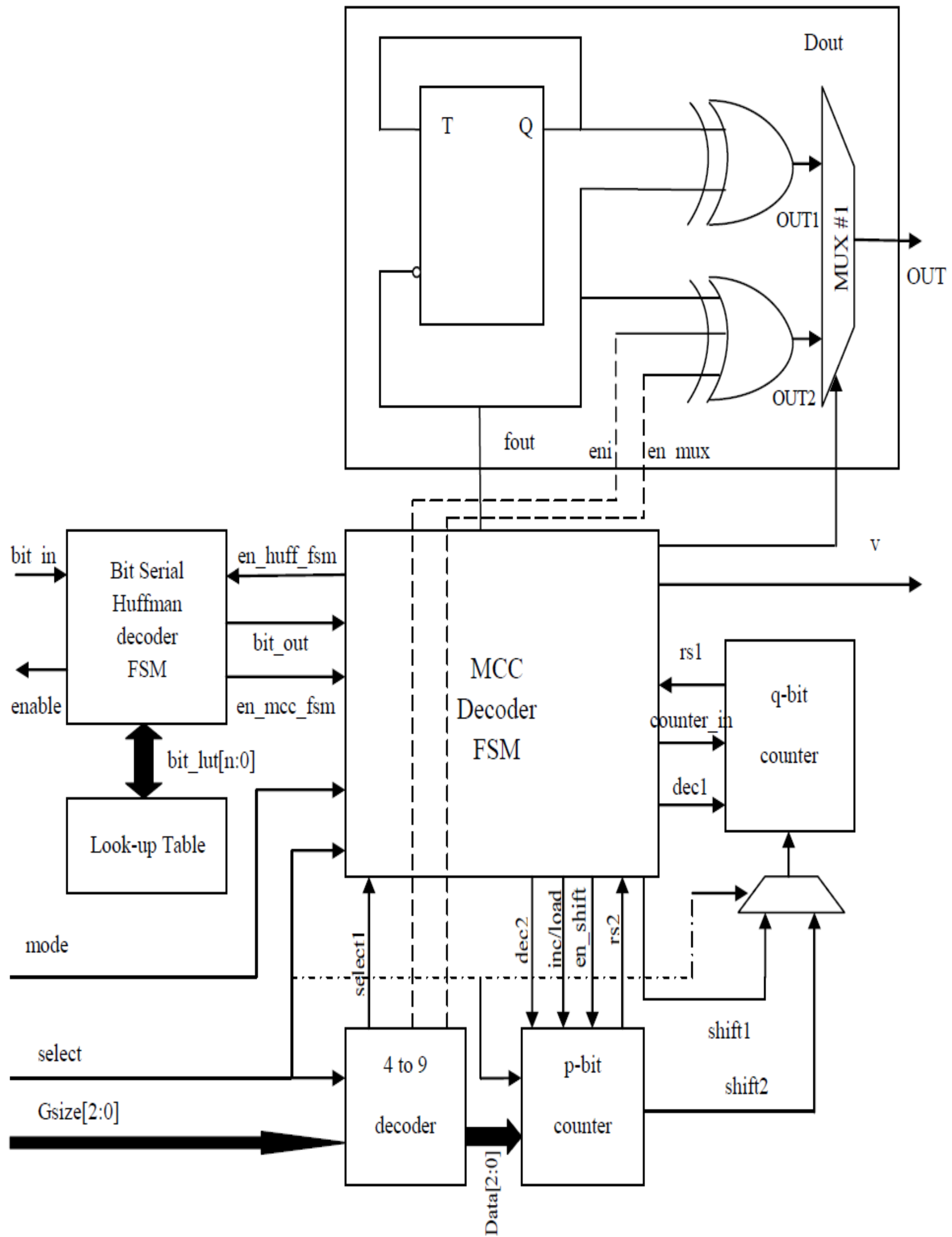


Figure 3.5: Architectural Diagram of Decompressor

integrated decompressor. The FSM block has 7 primary input pins, 1 clock pins, 1 synchronous reset pin and 3 primary output pins. The pin bit_out is used to receive data from the output of huffman decoder. The mode signal is used to determine decoding or bypass mode. The value of Gsize [2:0] signals can assign the parameter m or be used to recognize which decoding methods is used. The signal en_FSM use to enable the IIP and the pin out transfer decoded test data to core under test. The en pin is a commutation path between ATE and decoder to notice external tester can send next bit encoded data. The truth table of mode, select, and Gsize [2:0] is clearly defined the relationship between operation methodology and the value of these pins, shown in Table 3.4.

Mode	**Select**	**Gsize[2]**	**Gsize[1]**	**Gsize[0]**	**State**
0	0	X	X	X	Bypass
1	0	0	0	0	Golomb m = 2
1	0	0	0	1	Golomb m = 4
1	0	0	1	0	Golomb m = 8
1	0	0	1	1	Golomb m = 16
1	1	0	0	0	FDR
1	1	0	0	1	EFDR
1	1	1	1	1	AFDR

Table 3.4: The Truth Table of Control Signals

The 4-16 decoder has two purposes, one is to depend on signal of Gsize [2:0] pin to decode the data to send to p-bit counter, other is to decode control signals to blocks FSM and Dout. The truth table of the 4-16 decoder is shown in Table 3.5. While select = 0 and Gsize [2:0] = 000, the Golomb decoding mode with m =2, the signals d_{out} [4:0] send the data that is the binary value of m-1 to p-bit counter for preparing to count down, and the other signals are low.

Input Signals				State	Output signals			
Select	GSize[2]	GSize[1]	GSize[0]		Select 1	en_mux	en_i	Data [4:0]
0	0	0	0	Golomb m =2	0	0	0	00001
0	0	0	1	Golomb m =4	0	0	0	00011
0	0	1	0	Golomb m =8	0	0	0	00111
0	0	1	1	Golomb m =16	0	0	0	01111
1	0	0	0	FDR	1	0	0	00000
1	0	1	0	EFDR	0	0	0	00000
1	0	1	1	AFDR	1	1	0	00000

Table 3.5: The Truth Table of Control Signals for 4:16 Decoder

If select = 1, FDR similar decoding mode, the signals select 1, en_mux, and en_i output control signals with different decoding methods. The p-bit counter is not only used to count the prefix and tail length of codeword in FDR similar decoding, it but also parallel accept the binary value of m-1 from the 4-16 decoder and serial shift m-1 data into q-bit counter via path shift 2 and the multiplexer MUX#2. Moreover, the q-bit counter also accepts the prefix or tail from path counter_ in. For the FDR-similar decoding mode, we assume that the l_{max} is the longest run of 0's or 1's in test set T_D (or T_{diff}) and let $k = \lceil \log_2 l_{max} \rceil$. The value of k represents the number of bit of prefix (or tail) in codeword which the l_{max} has been encoded by FDR-similar methodology. It means that the size of k is changed with different requirements of test set. For the Golomb decoding mode, we know that parameter m is refereed to group size and the number of bit of tail in any group is $\log_2 m$, thus the value of m must be decided when we use Golomb code. However, we define $p = \max(\log_2 m, \log_2 k)$ and $q = \max(\log_2 m, k)$, it means that the value of p is the bigger one between $\log_2 K$ and $\log_2 m$,

and the q equal the bigger one between log_2m and k. The D_{out} block accept control signals from the 4-16 decoder and the FSM to output correct 0's or 1's run length. The FSM block consists of nine states to control the other block. The signals dec1 and dec2 are used to decrement, the signals rs1 and rs2 indicate the reset state of the counter, and the shift 1 and en_shift are the enable singles to indicate to shift data. The detail operation is as follows:

- **A.** In mode=1 and select= 0 , the Golomb decoding, while signal bit_in accept a 1, the FSM enable signal inc/load to become high and to notify p-bit counter to load the value of m-1 from 4-16 decoder through path data [4:0]. The p-bit counter counts down to zero. The signal en is low when the p-bit counter is busy with counting and enables the input at the end of m cycle to accept another bit. During this operation, the decoder outputs m zeros via out and make the valid signal v to become high.
- **B.** When the input is zero, the FSM starts decoding the tail of the input codeword. At the same time, the signal inc/load becomes high to enable p-bit counter for parallel loading the value of m -1. In the next cycle, the m-1 serial shift into q-bit counter via path shift2. This operation is a transformation of data [4:0] that from data type to control signal type. If m =4, the number of bit of tail is 2 bits , see Table 5-1, this mean q-bit counter needs two enable signals to fetch two bits data of tail in two clock cycles. This, data [4:0] = 00011 provide two bits of 1 to enable q bit counter.
- **C.** When mode = 1, and select=1, the FDR similar decoding, the FSM supply the q-bit counter with the prefix. The end of the prefix is separated by the zero. The signals en, shift 1 and inc/load are high until the 0 is received. In this operation, the q-bit counter is used to store the data of prefix, and the p-bit counter uses for counting the number of bit of prefix.
- **D.** The FSM outputs zeros, decrements the q-bit counter, and makes the signal dec1 high. It continues to output zeros throughout till rs1 become high, p-bit counter is decremented, and the signal rs2 indicates while it is in the zero throughout till rs1 becomes high. The signal v is used to identify a valid output.
- **E.** The tail part shifted in q– bit counter through path counter in unit the p-bit counter counts down to zero. The signal dec2 maintains while it is in the zero state.
- **F.** The FSM output 0s depending on the tail followed by a 1 at the end of tail decoding.

3.2.2 Finite State Machine Representation of Decompressor

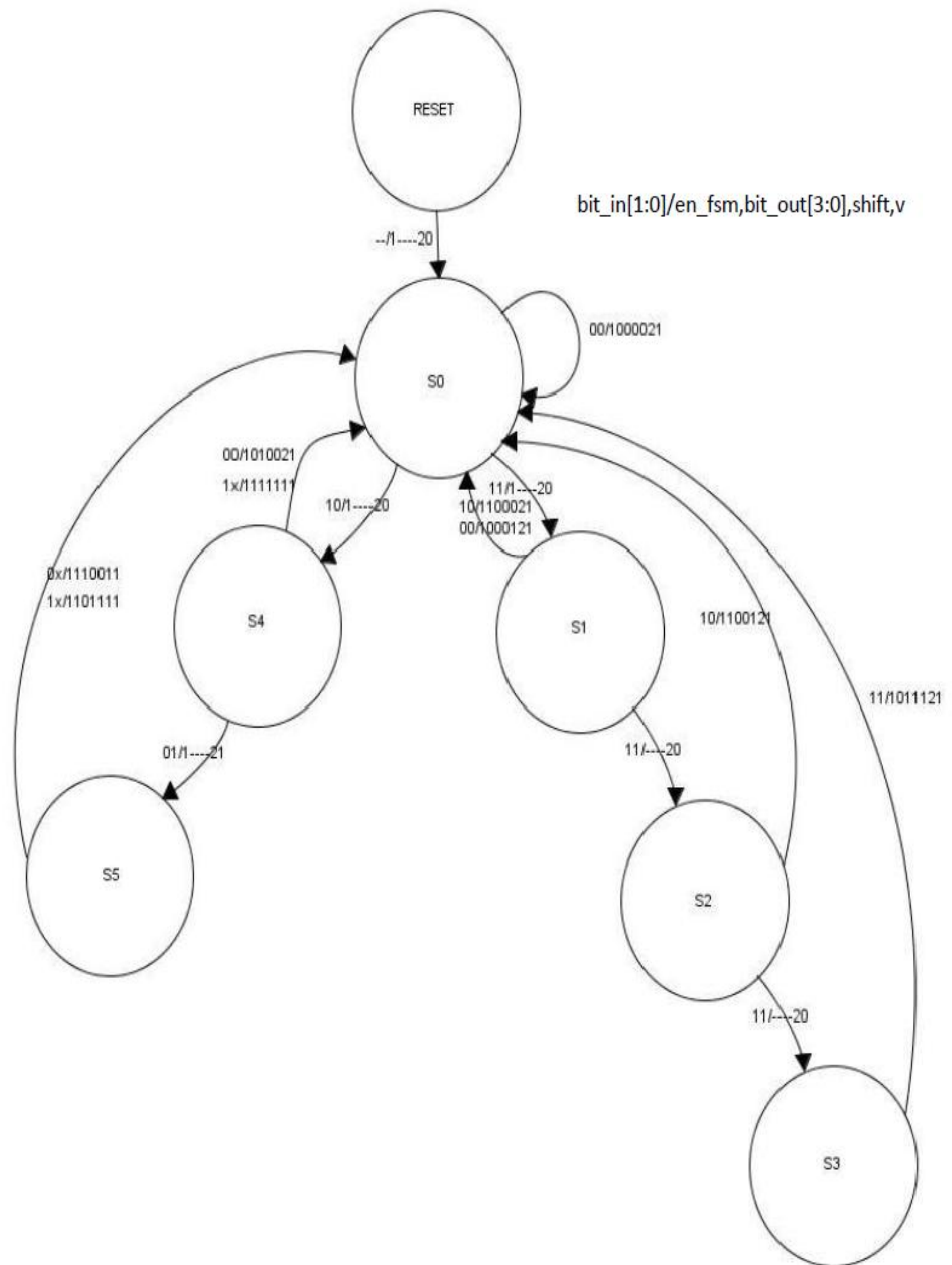


Figure 3.6: State Diagram of 2 bit parallel Huffman Decoder

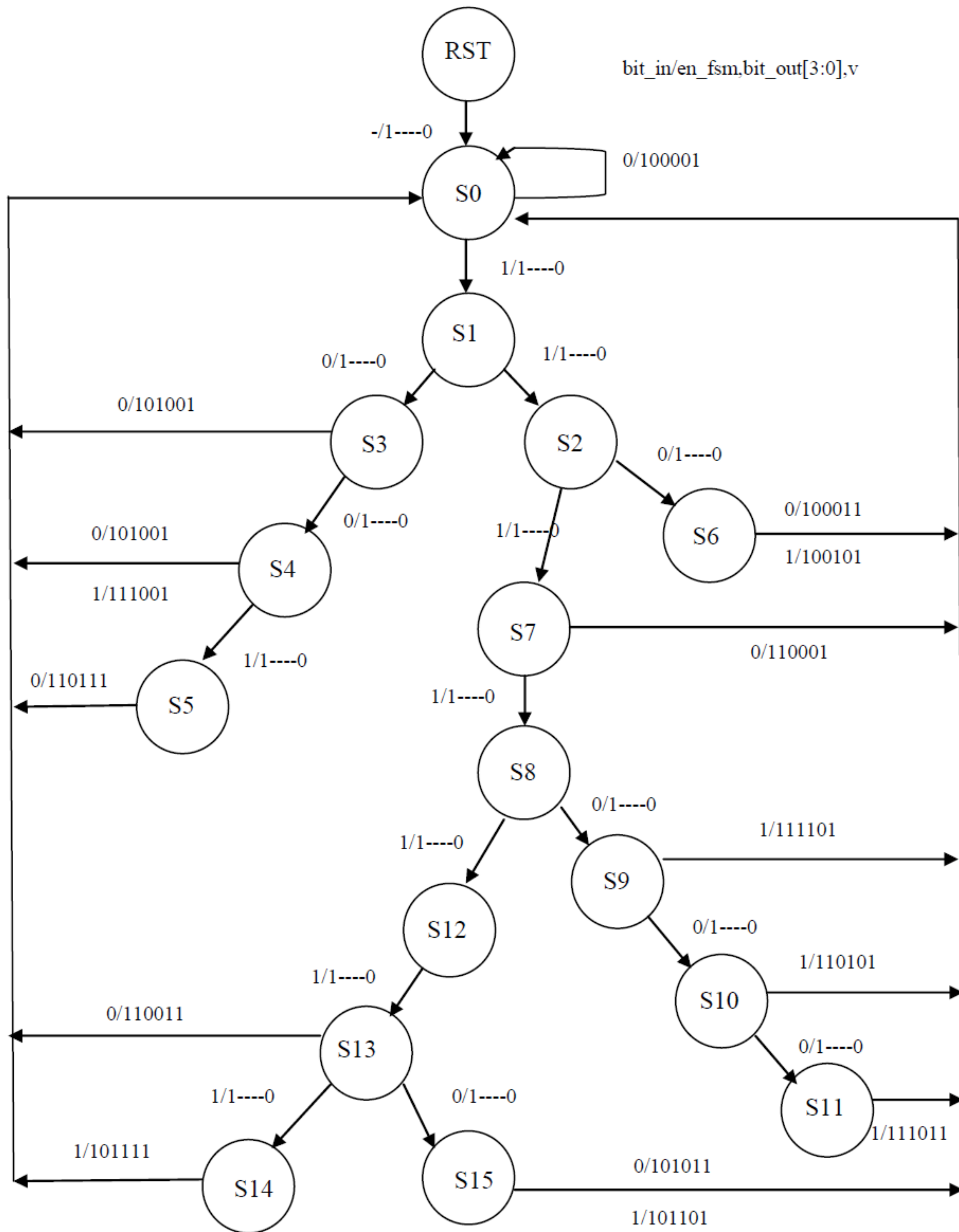


Figure 3.7: State Diagram of bit serial Huffman Decoder

bit_in,rs1,rs2,select,select1,en_FSM,mode/fout,v,en,counter_in,shift1,dec1,inc,dec2,enshift,enmux

S0
-----0-/-01-000000
-----10/-01-000000
-----0-/-01-000000
-----10/-11-000000
S1
0----11/-01-001000
1—0011/010-001000
1—1111/-01110100-
---1011/-00000000-
--00011/010-000100
S2
1—1011/-01110100-
1—1111/-01110100-
S3
1—0011/010-001000
0—1011/-01-00100-
0—0111/-00010100-
--10011/-01-000000
S4
S5
-0-0111/010-01000-
-1-0111/-01-000000-
0-00011/-01-001000
S6
--0—11/-01110011-
S7
--0—11/-00-00000-
--1—11/-00-00000-
-0---11/111-00000-
-1---11/111-00000-
-1---0-/111-00000-
S8

Figure 3.8: State Diagram of Integrated Decompressor of Run Length Based Codes

3.3 X-Compaction for output response compaction

X-compaction [7] scheme can be applied then to the response obtained when the decompressed vectors are applied to CUT. The architecture of this technique was already discussed in the previous chapter. In this way volume of test data can be reduced significantly both at the input and output side in order to reduce the testing time considerably and hence the testing cost which depends on the testing time can be considerably reduced.

3.4 Tools Used

1. Visual C++: It has been utilized for implementing various compression schemes, and for implementing Double Hamming Distance Based Reordering Technique.

2. DESIGN VISION: It has been used for implementing Scan chain insertion and DFT and for synthesizing the Decoder.

3. TETRAMAX: It has been used as an ATPG for generating Test vectors for four ISCAS 89 Benchmark circuits.

CHAPTER 4 – Results and Analysis

Several experiments have been conducted in order to implement various compression schemes. The goal with which these experiments have been performed show the importance of test data compression algorithms in minimizing testing time. In this chapter the results of various experiments conducted on various ISCAS89 benchmark circuits are presented. C programming has been used for implementing various compression schemes, Double Hamming Based Reordering technique, bit stuffing techniques .

SCAN based technique has been used for generating test vectors from TETRAMAX which works on the fact of making each flip flop in the circuit directly controllable by making it primary input during testing.

Also results from DESIGN VISION which is used for SCAN insertion and DFT for ISCAS benchmark circuits are also presented for Four ISCAS 89 circuits which show how the circuits are synthesized.

Synthesis Results on Design Compiler

Synopsys DC tool version D-2010.03-SP1 is used for synthesis. The Working Environment is :

- Operating Condition Name : WCCOM
- Library : fsd0c_a_generic_core_ss1p08v125c
- Process : 1.00
- Temperature : 125.00 0C
- Voltage : 1.08 V
- Interconnect Model : Wire load (worst case tree)
- Technology: 90 nm
- Timing Analysis Effort : Medium
- Power Analysis Effort : Low
- Wire Load Model : G10K

4.1 Results of test compression algorithms using C programming

In this section, experimental results of test data compression algorithms using C Programming method for four ISCAS 89 circuits have been presented.

4.1.1 Circuit S298

Circuit	Compression			
S298	**Golomb**	**FDR**	**EFDR**	**AFDR**
	86.2	90.26	89.12	88.96

Table 4.1: Results for ISCAS'89 S298 circuit using RunLength based compression code for TETRAMAX ATPG patterns

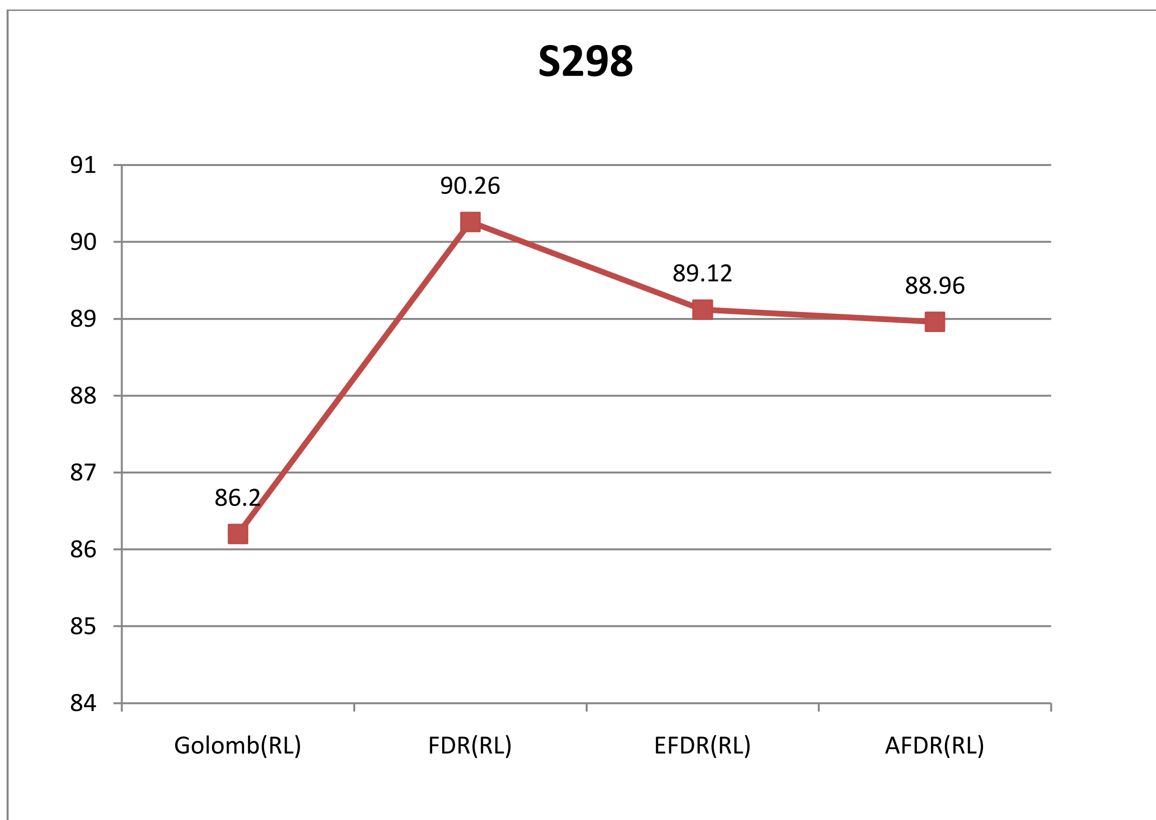


Figure 4.1: Summary of ISCAS'89 S298 circuit using RunLength based compression code

Circuit S298	Compression	
	Run Length	**Run Length + Huffman**
	68.87	71.6

Table 4.2: Results for ISCAS'89 S298 circuit using RunLength and proposed RunLength + Huffman compression code for TETRAMAX ATPG patterns

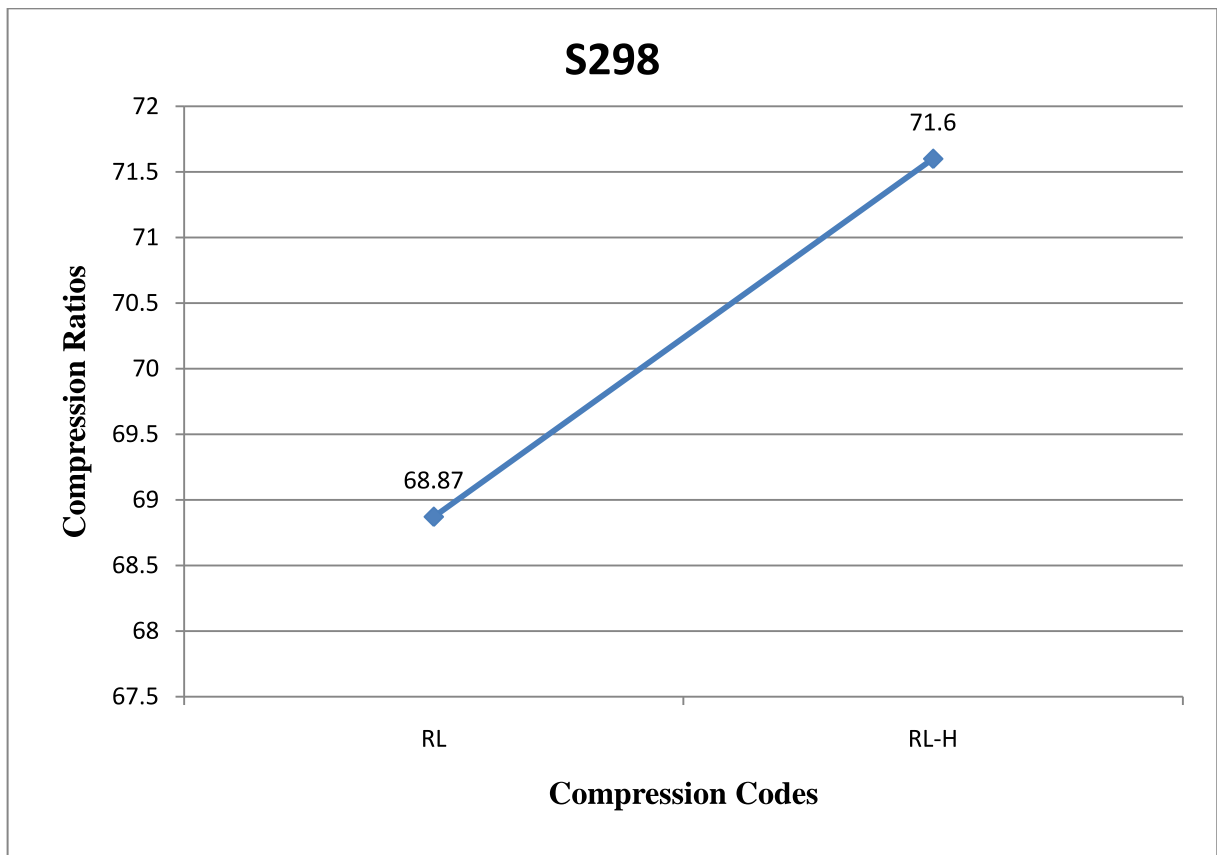


Figure 4.2: Summary of ISCAS'89 S298 circuit using RunLength and proposed RunLength + Huffman compression code

Circuit S298				
	Compression			
	Golomb (RL+4H)	**FDR (RL+4H)**	**EFDR (RL+4H)**	**FDR (RL+4H)**
	90.09	92.74	92.5	93.7

Table 4.3: Results for ISCAS'89 S298 circuit using RunLength + Huffman (m=4)for TETRAMAX ATPG patterns

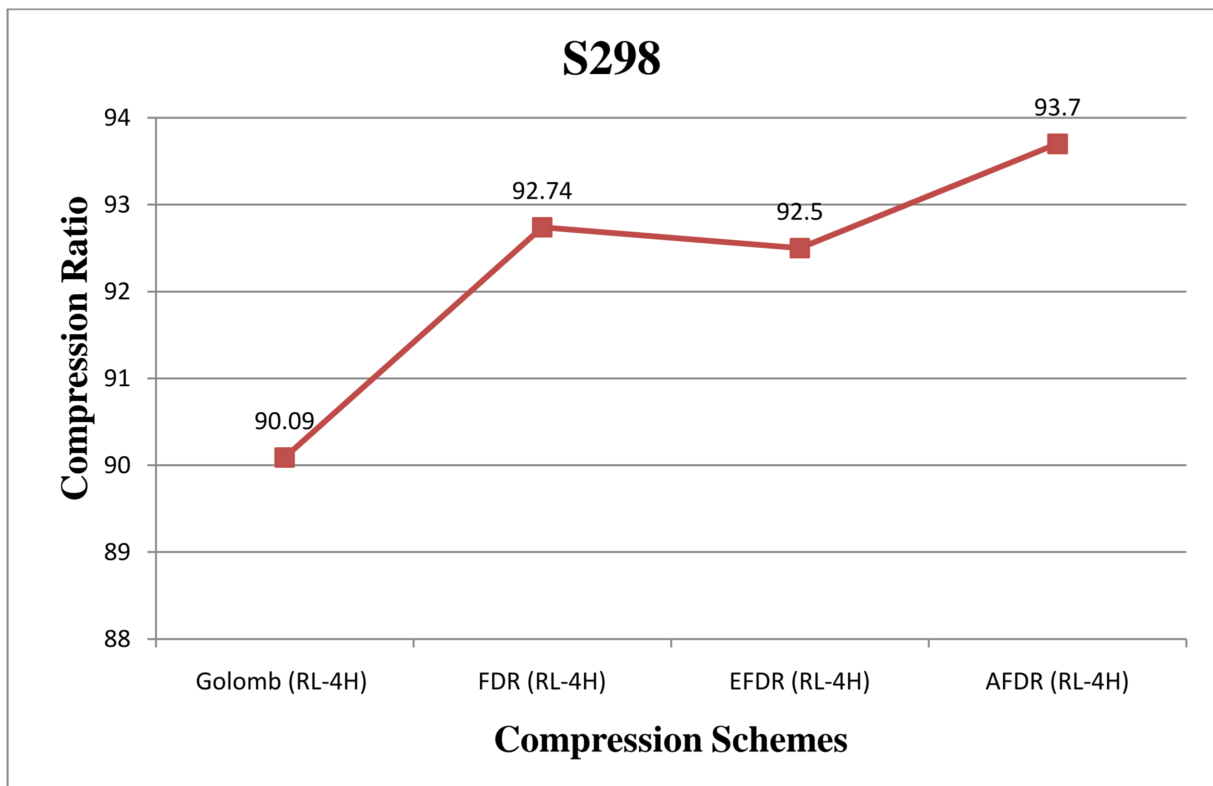


Figure 4.3: Summary of ISCAS'89 S298 circuit using RunLength + Huffman (m=4)

Circuit	Compression			
S298	**Golomb (RL+5H)**	**FDR (RL+5H)**	**EFDR (RL+5H)**	**AFDR (RL+5H)**
	93.07	93.6	92.8	93.9

Table 4.4: Results for ISCAS 89 S298 circuit using RunLength + Huffman (m=5) for TETRAMAX ATPG patterns

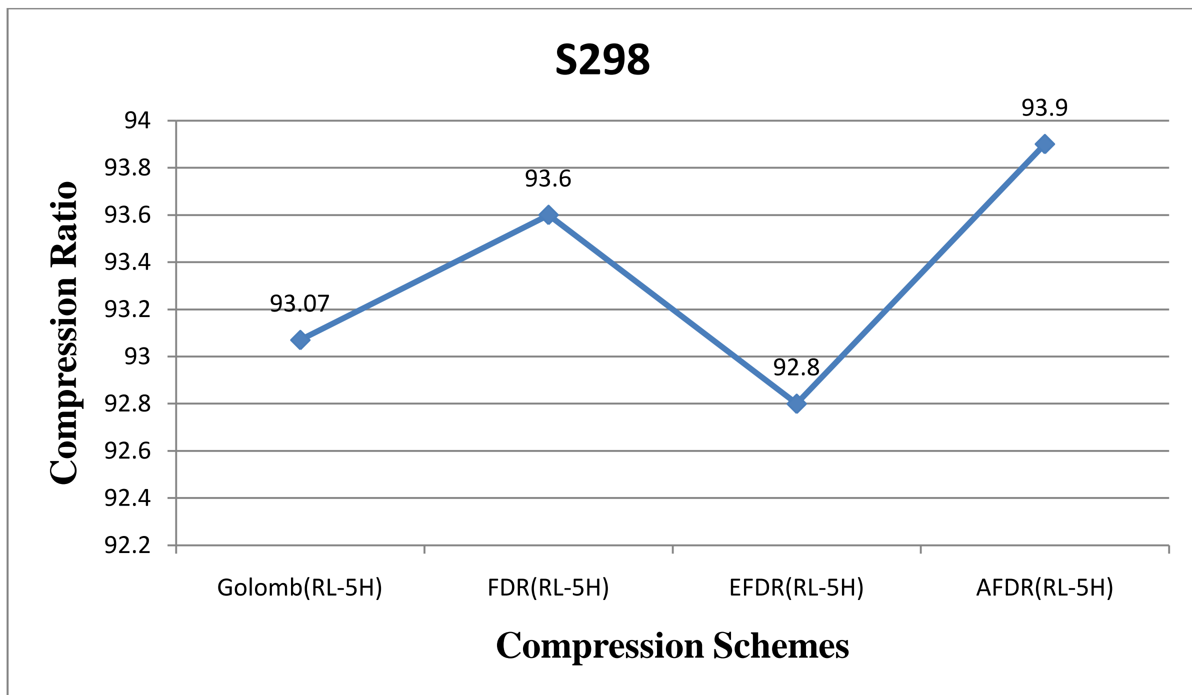


Figure 4.4: Summary of ISCAS'89 S298 circuit using RunLength + Huffman (m=5).

4.1.2 Circuit S400

Circuit	Compression			
S400	**Golomb**	**FDR**	**EFDR**	**AFDR**
	91.02	94.01	93.14	93.27

Table 4.5: Results for ISCAS'89 S400 circuit using RunLength based compression code for TETRAMAX ATPG patterns

Figure 4.5: Summary of ISCAS'89 S400 circuit using RunLength based compression code

Circuit	Compression	
S400	**Run Length**	**Run Length + Huffman**
	57.4	67.4

Table 4.6: Results for ISCAS'89 S400 circuit using RunLength and proposed RunLength + Huffman compression code for TETRAMAX ATPG patterns

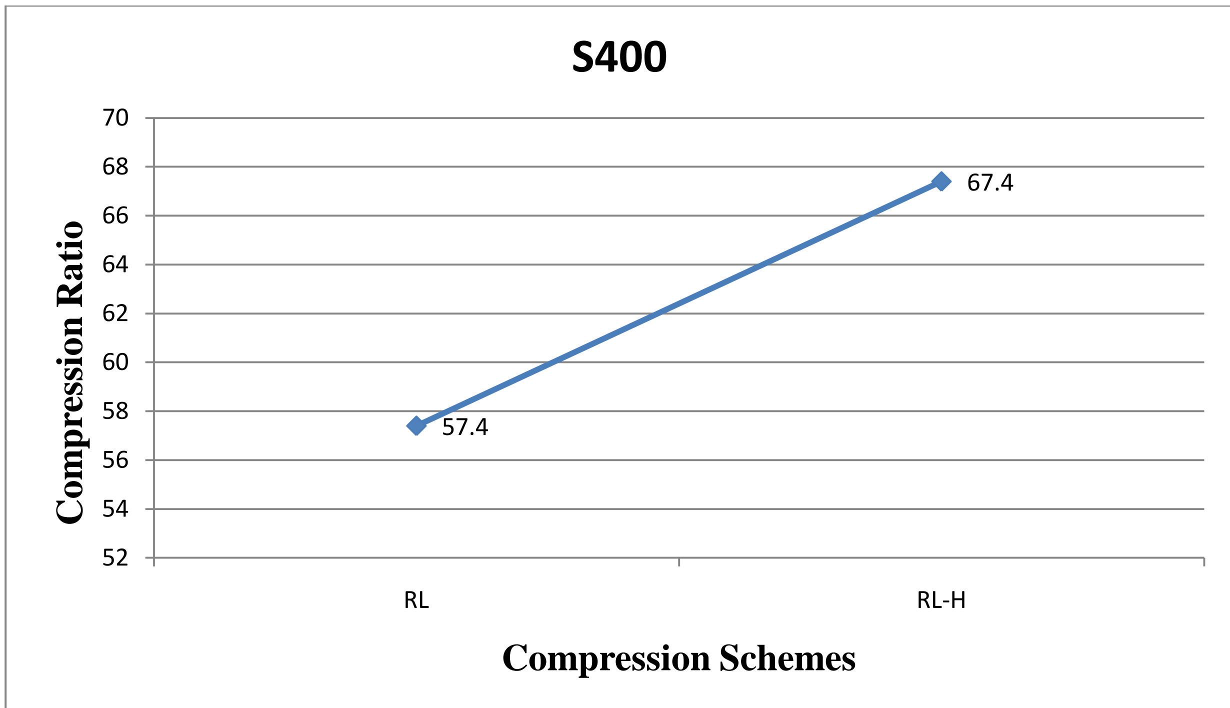


Figure 4.6: Summary of ISCAS'89 S400 circuit using RunLength and proposed RunLength + Huffman compression code

Circuit	Compression			
S400	Golomb (RL+4H)	FDR (RL+4H)	EFDR (RL+4H)	AFDR (RL+4H)
	94.76	95.3	94.45	94.38

Table 4.7: Results for ISCAS 89 S400 circuit RunLength + Huffman (m=4) for TETRAMAX ATPG patterns

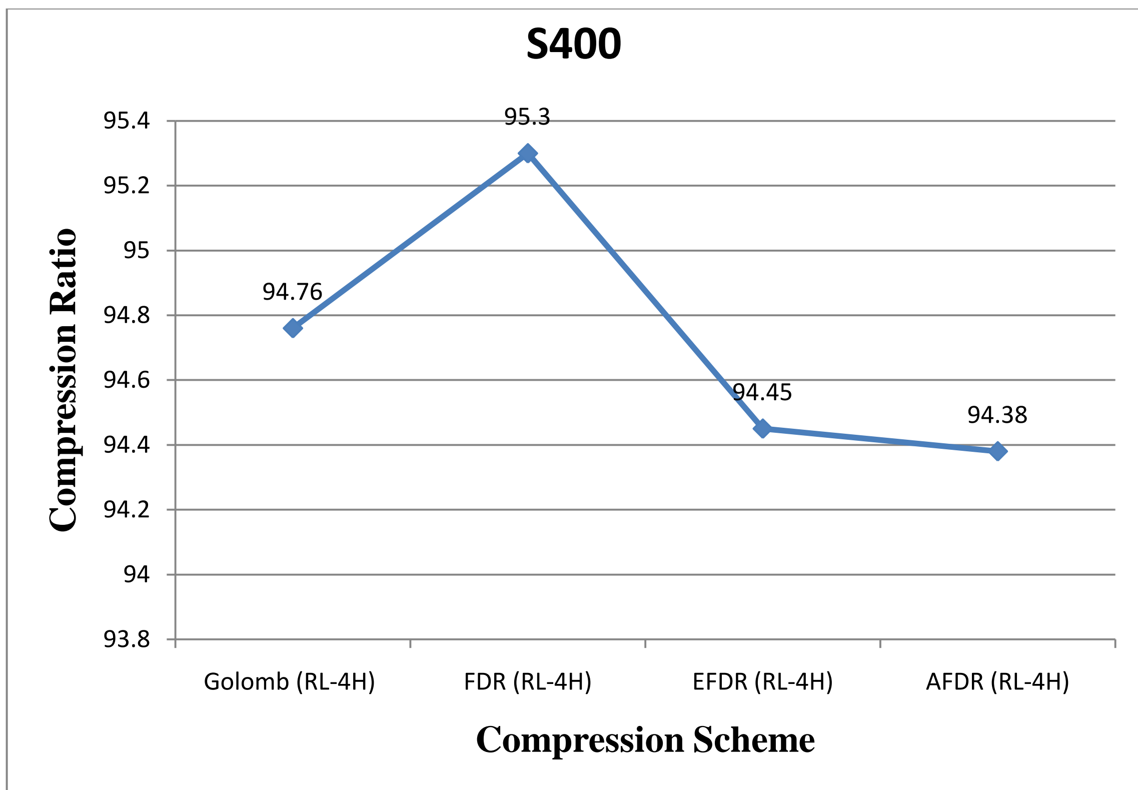


Figure 4.7: Summary of ISCAS’89 S400 circuit RunLength + Huffman (m=4)

Circuit	**Compression**			
S298	**Golomb (RL+5H)**	**FDR (RL+5H)**	**EFDR (RL+5H)**	**AFDR (RL+5H)**
	95.12	94.04	96.3	94.13

Table 4.8: Results for ISCAS'89 S400 circuit using RunLength + Huffman (m=5) for TETRAMAX ATPG patterns

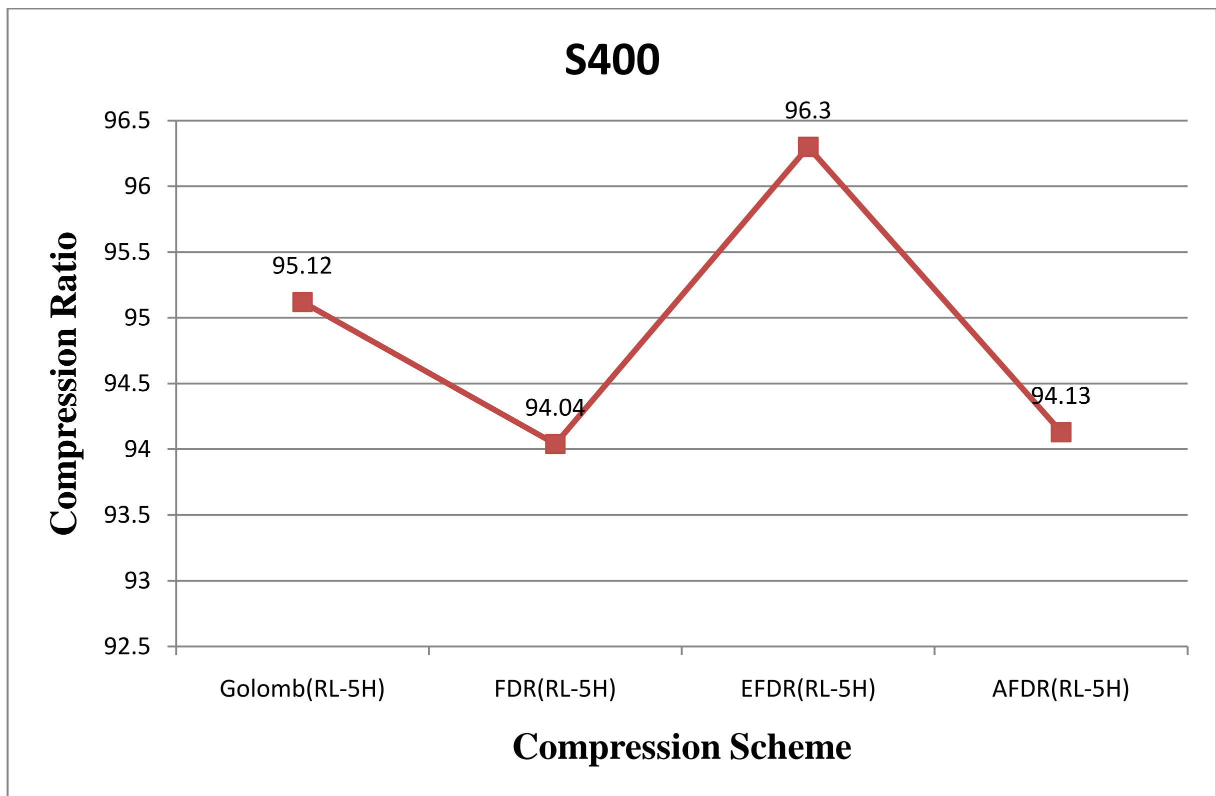


Figure 4.8: Summary of ISCAS'89 S400 circuit using RunLength + Huffman (m=5)

4.1.3 Circuit S1494

Circuit	Compression			
S1494	**Golomb**	**FDR**	**EFDR**	**AFDR**
	88.18	89.27	88.83	89.27

Table 4.9: Results for ISCAS'89 S1494 circuit using RunLength based compression code for TETRAMAX ATPG patterns

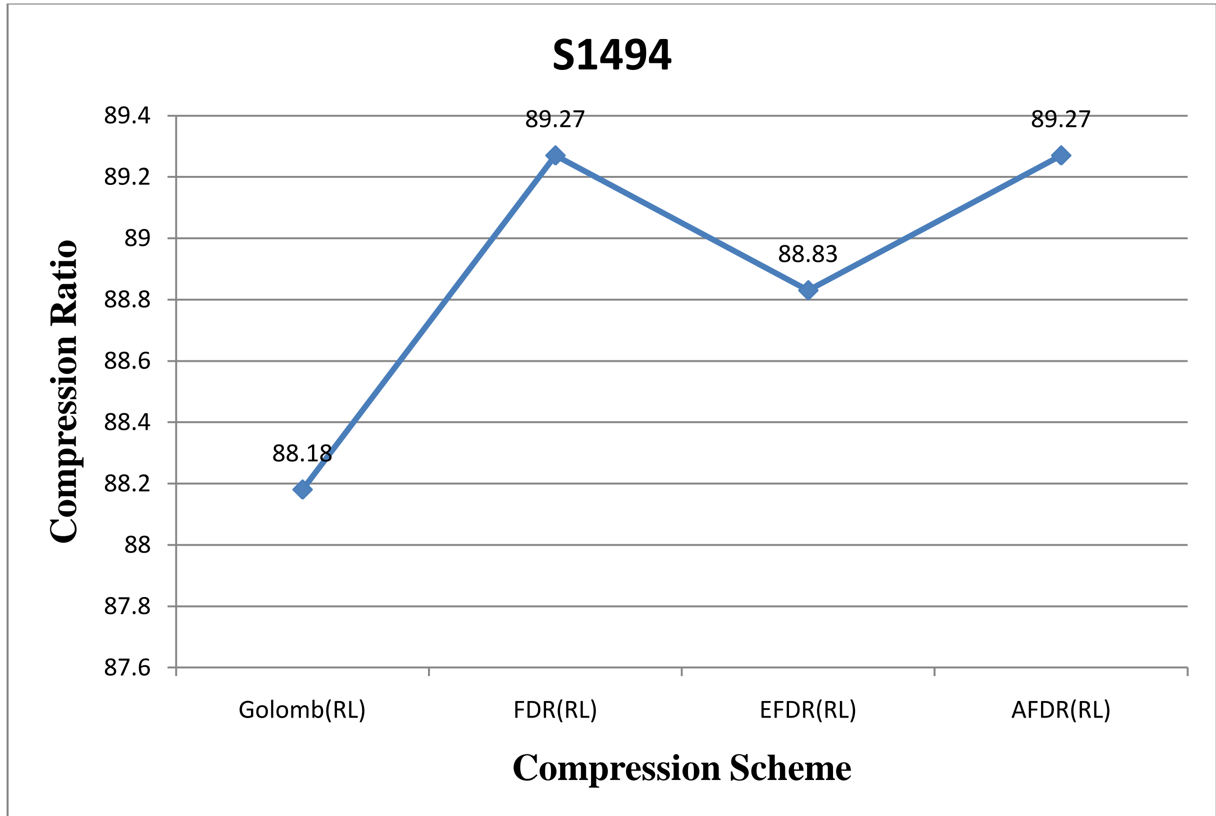


Figure 4.9: Summary of ISCAS'89 S1494 circuit using RunLength based compression code

Circuit	Compression	
S1494	Run Length	Run Length + Huffman
	77.7	82.55

Table 4.10: Results for ISCAS'89 S1494 circuit using RunLength and proposed RunLength + Huffman compression code for TETRAMAX ATPG patterns

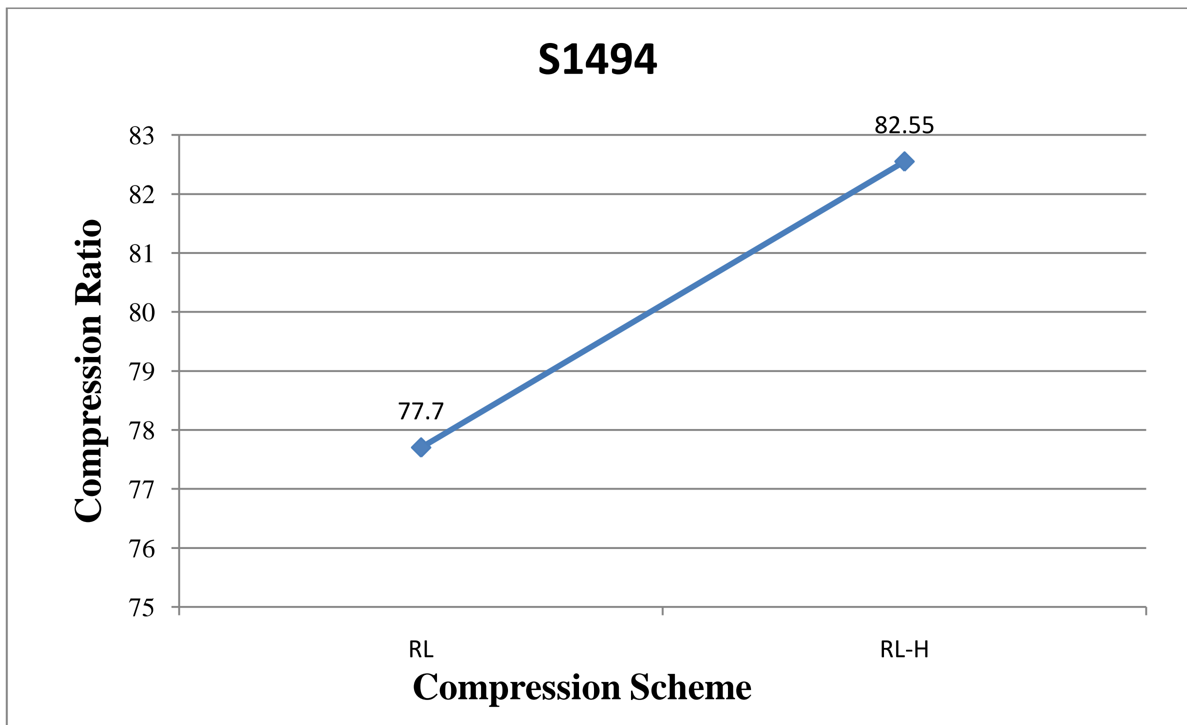


Figure 4.10: Summary of ISCAS'89 S1494 circuit using RunLength and proposed RunLength + Huffman compression code

Circuit S1494	**Compression**			
	Golomb (RL+4H)	**FDR (RL+4H)**	**EFDR (RL+4H)**	**AFDR (RL+4H)**
	90.09	92.74	92.5	93.7

Table 4.11: Results for ISCAS'89 S1494 circuit RunLength + Huffman (m=4) for TETRAMAX ATPG patterns

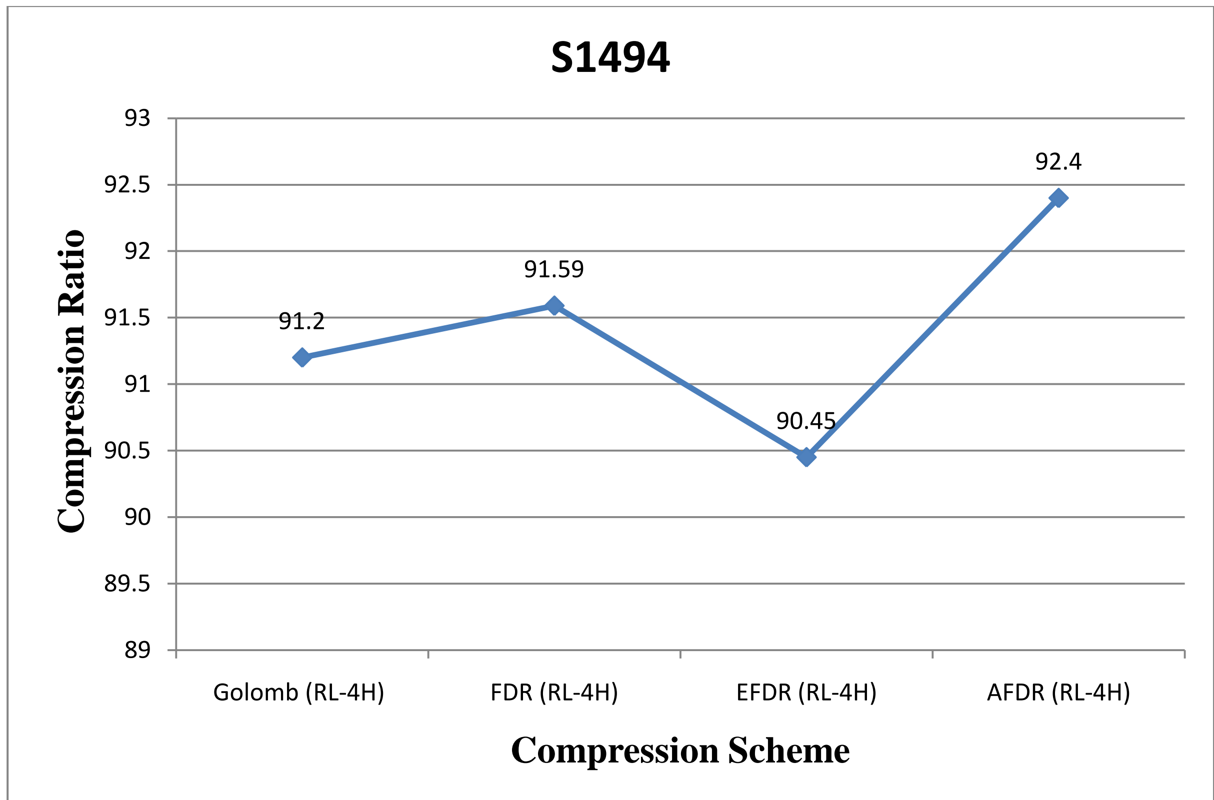


Figure 4.10: Summary of ISCAS'89 S400 circuit RunLength + Huffman (m=4)

Circuit	Compression			
S1494	Golomb (RL+5H)	FDR (RL+5H)	EFDR (RL+5H)	AFDR (RL+5H)
	94.35	93.4	92.14	93.5

Table 4.11: Results for ISCAS'89 S1494 circuit RunLength + Huffman (m=5) for TETRAMAX ATPG patterns

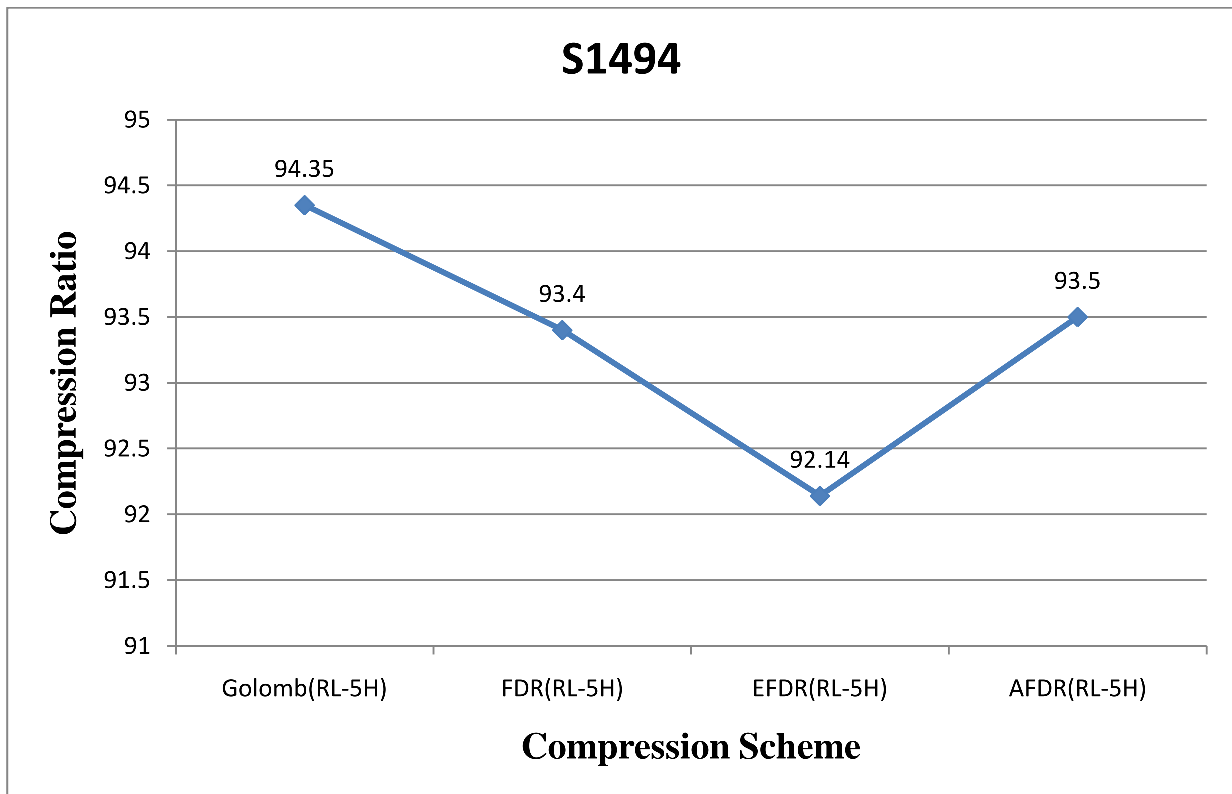


Figure 4.11: Summary of ISCAS'89 S1494 circuit RunLength + Huffman (m=5)

4.1.4 Circuit S1196

Circuit	Compression			
S1196	**Golomb**	**FDR**	**EFDR**	**AFDR**
	62.56	61.74	56.01	54.52

Table 4.12: Results for ISCAS'89 S1196 circuit using RunLength based compression code for TETRAMAX ATPG patterns

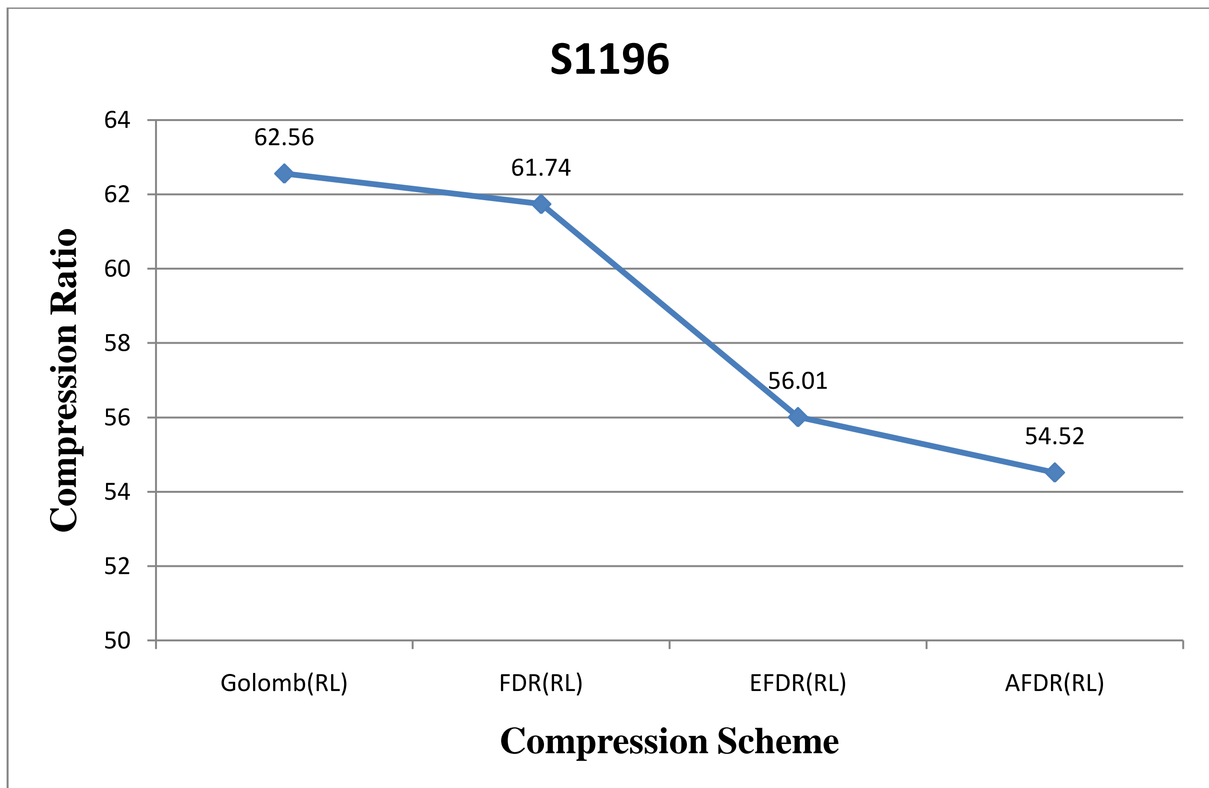


Figure 4.12: Summary of ISCAS'89 S1196 circuit using RunLength based compression code

Circuit	Compression	
S1196	**Run Length**	**Run Length + Huffman**
	70.12	77.4

Table 4.13: Results for ISCAS'89 using RunLength and proposed RunLength + Huffman compression code S1196 circuit for TETRAMAX ATPG patterns

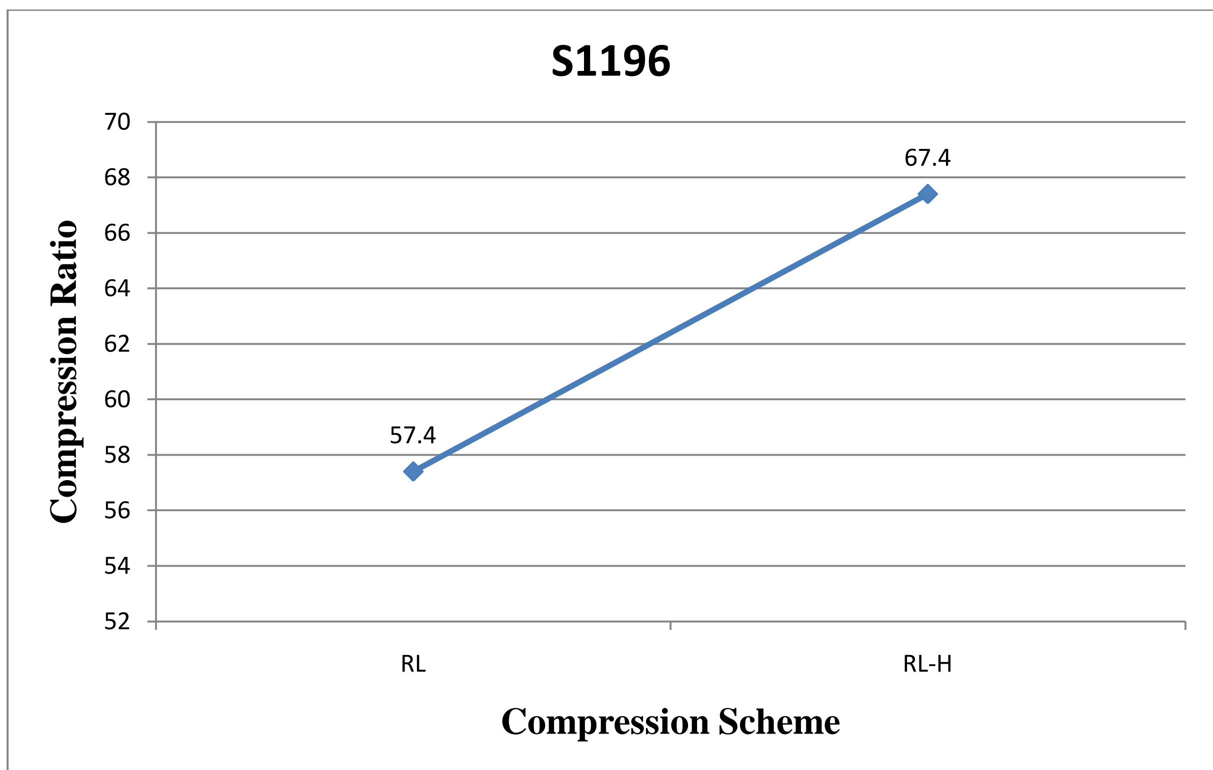


Figure 4.13: Summary of ISCAS 89 S1196 circuit using RunLength and proposed RunLength + Huffman compression code

Circuit	Compression			
S1196	Golomb (RL+4H)	FDR (RL+4H)	EFDR (RL+4H)	AFDR (RL+4H)
	66.17	66.4	60.05	59.11

Table 4.14: Results for ISCAS 89 S1196 circuit RunLength + Huffman (m=4) for TETRAMAX ATPG patterns

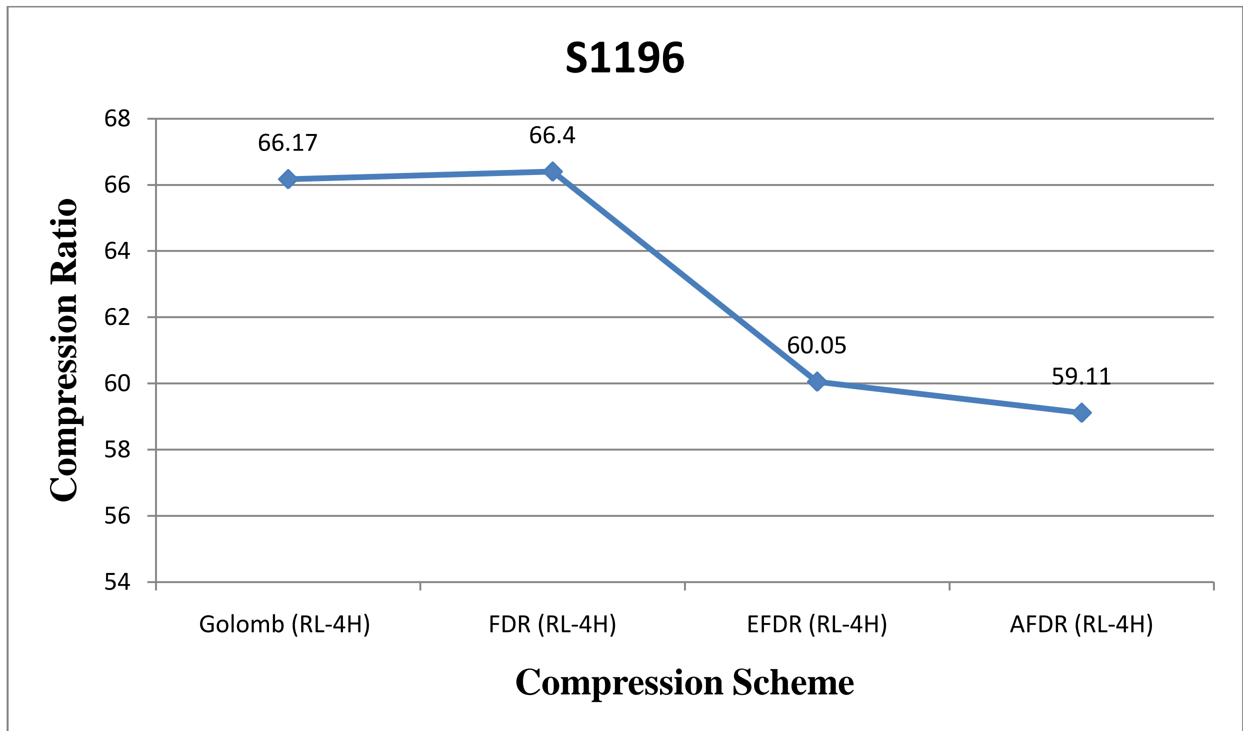


Figure 4.14: Summary of ISCAS'89 S1196 circuit RunLength + Huffman (m=4)

Circuit	Compression			
S1196	Golomb (RL+5H)	FDR (RL+5H)	EFDR (RL+5H)	AFDR (RL+5H)
	69.31	70.03	65.08	63.73

Table 4.15: Results for ISCAS'89 S1196 circuit RunLength + Huffman (m=5) for TETRAMAX ATPG patterns

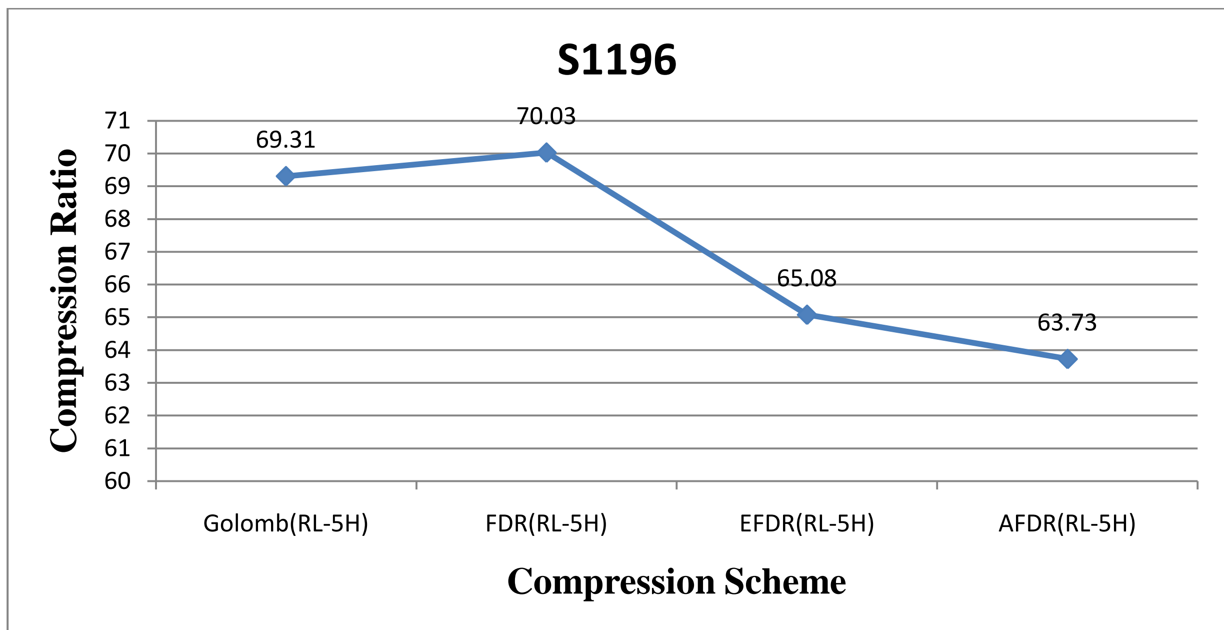


Figure 4.15: Summary of ISCAS'89 S1196 circuit RunLength + Huffman (m=5)

4.1.5 Comparison of results for various Compression Schemes

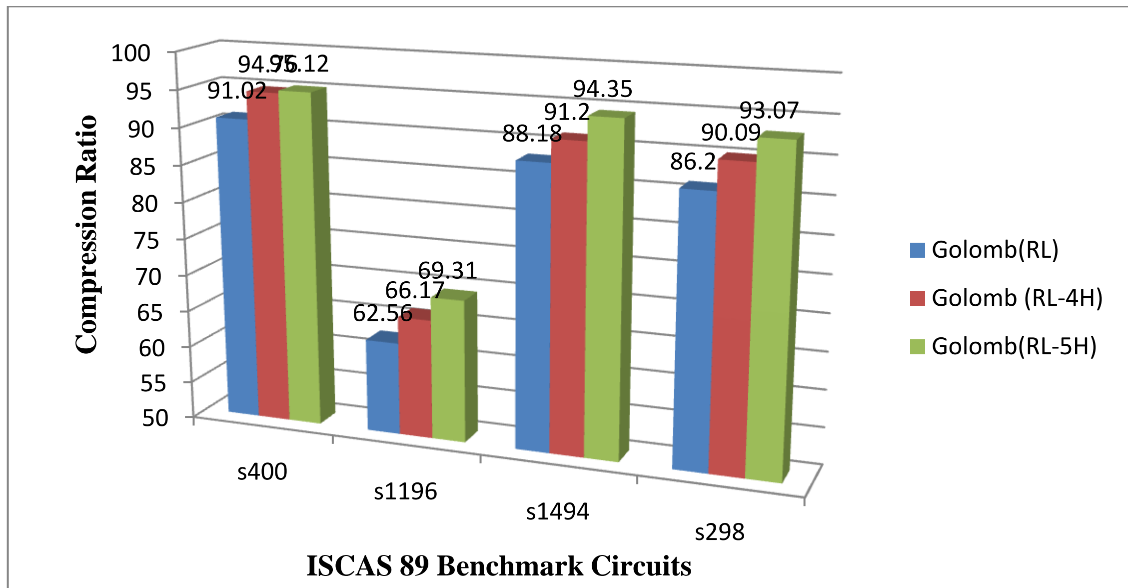


Figure 4.16 Comparison of compression ratios of Golomb and Golomb plus Huffman Coding Scheme for ISCAS 89 Benchmark Circuits

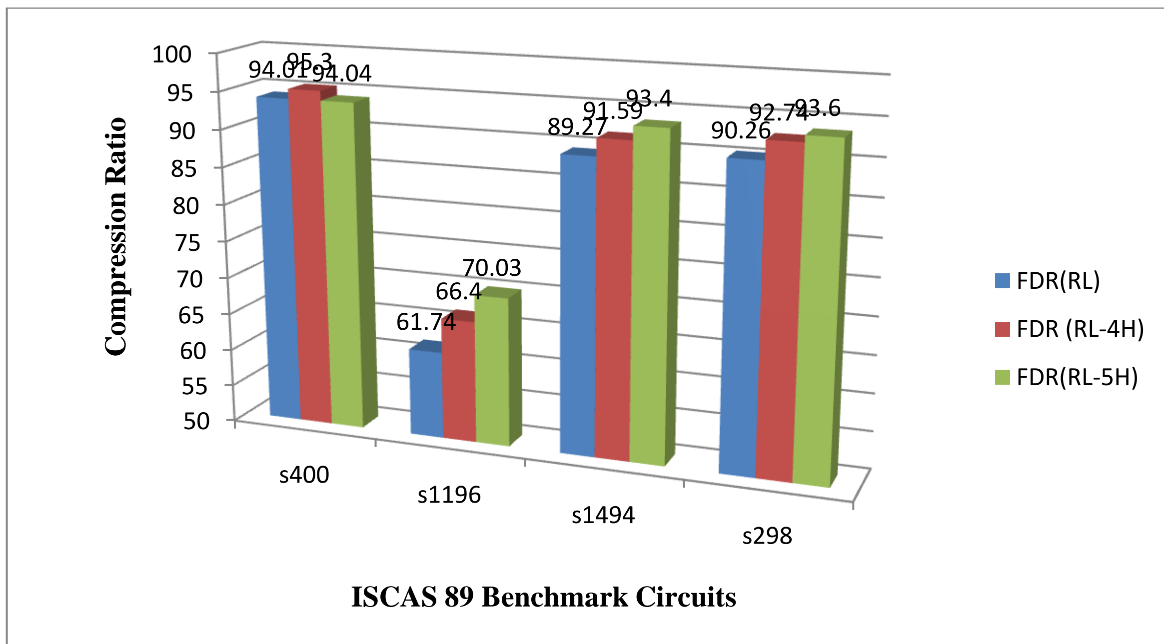


Figure 4.17 Comparison of compression ratios of FDR and FDR plus Huffman Coding Scheme for ISCAS 89 Benchmark Circuits

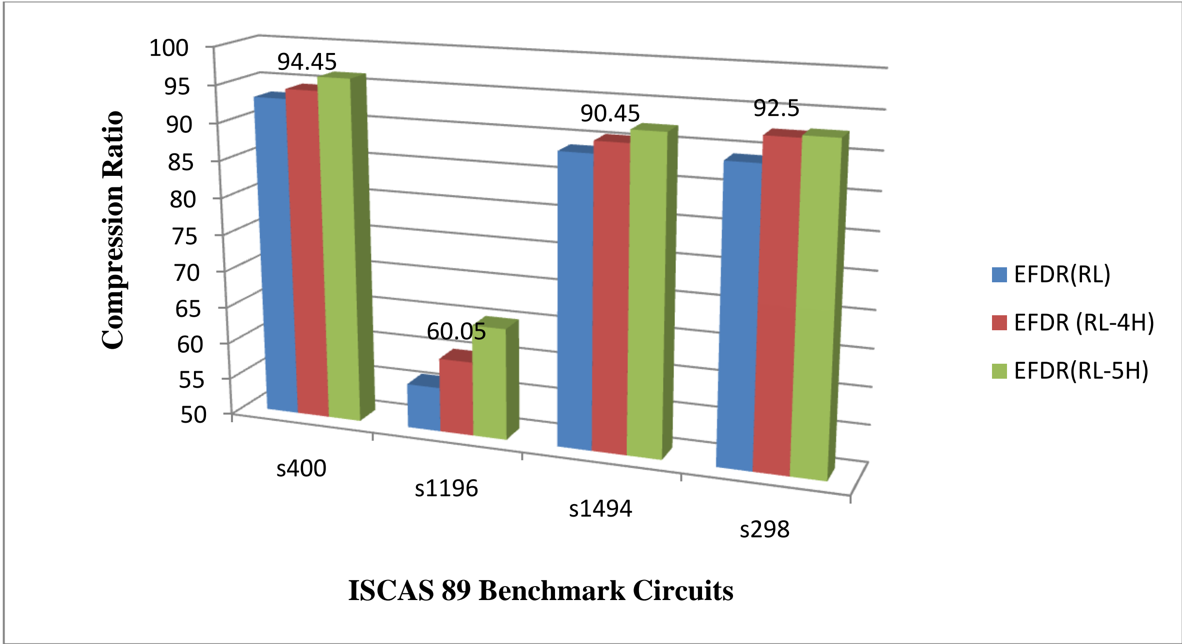


Figure 4.17 Comparison of compression ratios of EFDR and EFDR plus Huffman Coding Scheme for ISCAS 89 Benchmark Circuits

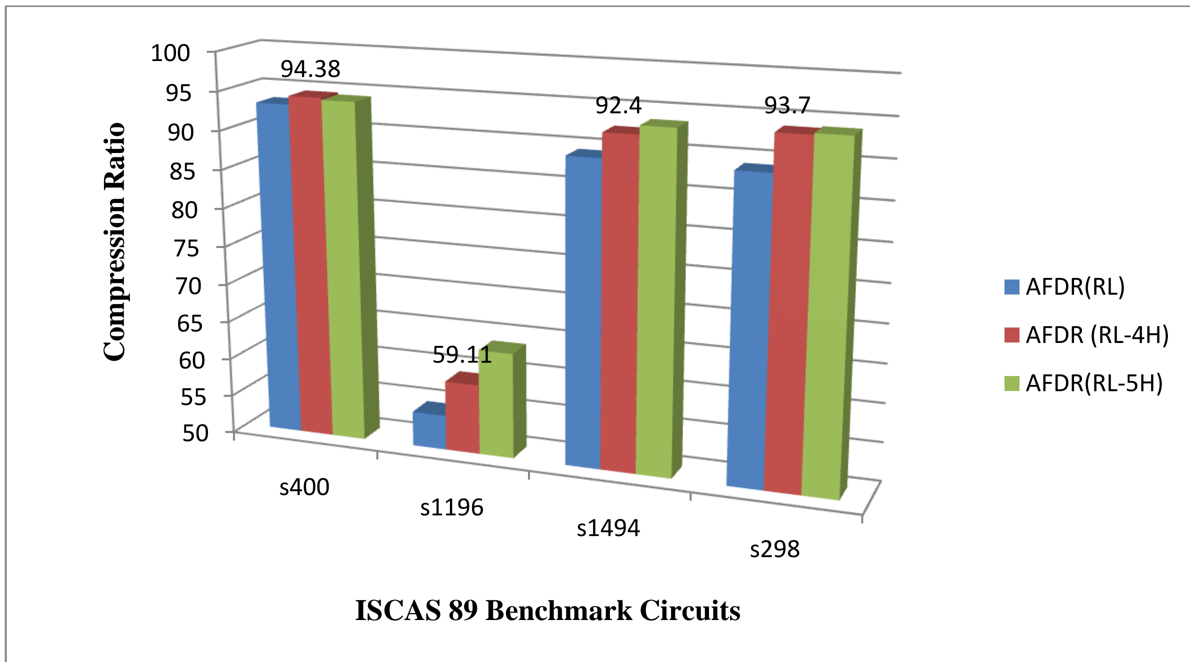


Figure 4.18 Comparison of compression ratios of AFDR and AFDR plus Huffman Coding Scheme for ISCAS 89 Benchmark Circuits

4.2 Design Vision Results of Decompressor

The decompressor has been implemented using VERILOG HDL (Hardware Description Language) ,simulated on MODELSIM (MENTOR GRAPHICS) and synthesized on DESIGN VISION (SYNOPSYS) .

The results for area, power and cell count are shown for Design vision.

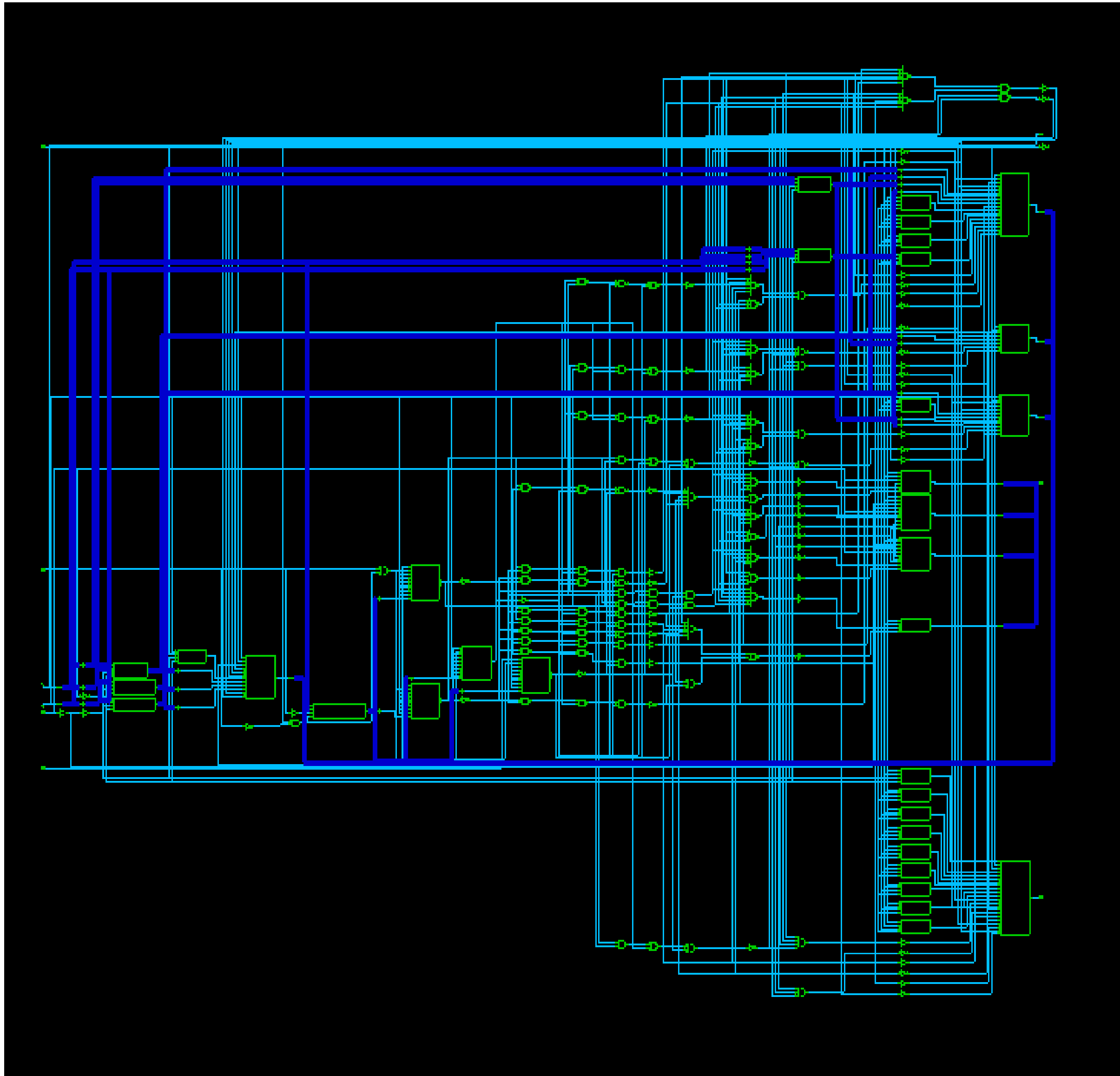


Figure 4.19: RTL of bit serial FSM based Huffman Decoder for ISCAS'89 S400 circuit

Combinational area:1215.546997 um^2

Noncombinational area:312.480011 um^2

Total cell area:1528.027222 um^2

Dynamic Power Units = 1mW (derived from V,C,T units)

Leakage Power Units = 1pW

Cell Internal Power = 3.7660 uW (63%)

Net Switching Power = 2.2336 uW (37%)

Total Dynamic Power = 5.9996 uW (100%)

Cell Leakage Power = 5.9190 nW

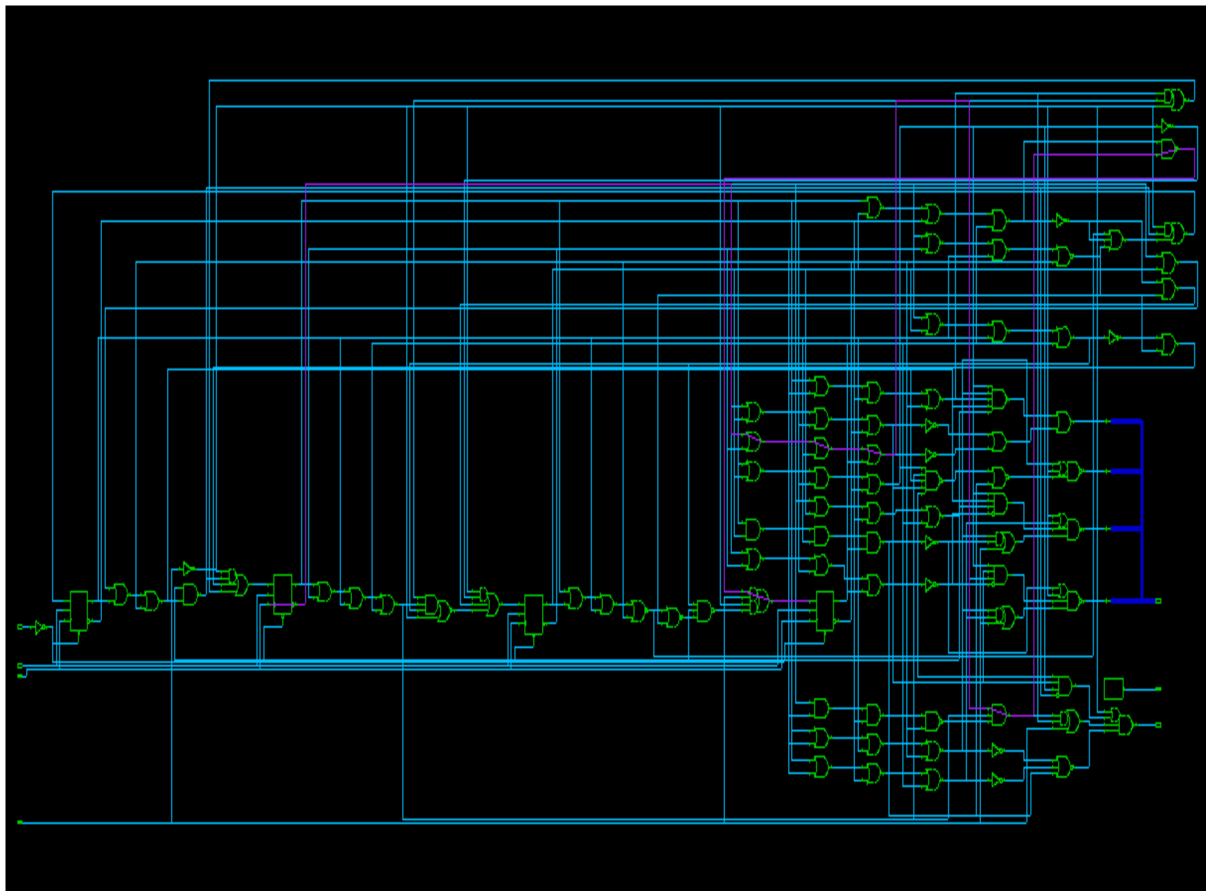


Figure 4.20: Bit serial FSM based Huffman Decoder for ISCAS'89 S400 circuit after synthesis

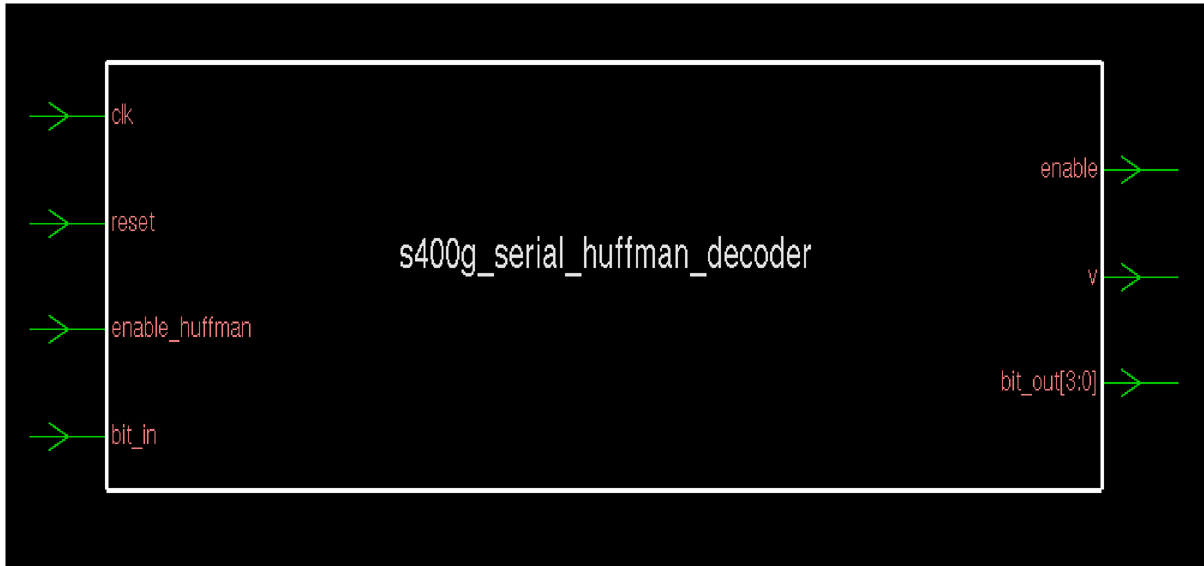


Figure 4.21: Block Diagram of Serial Huffman Decoder

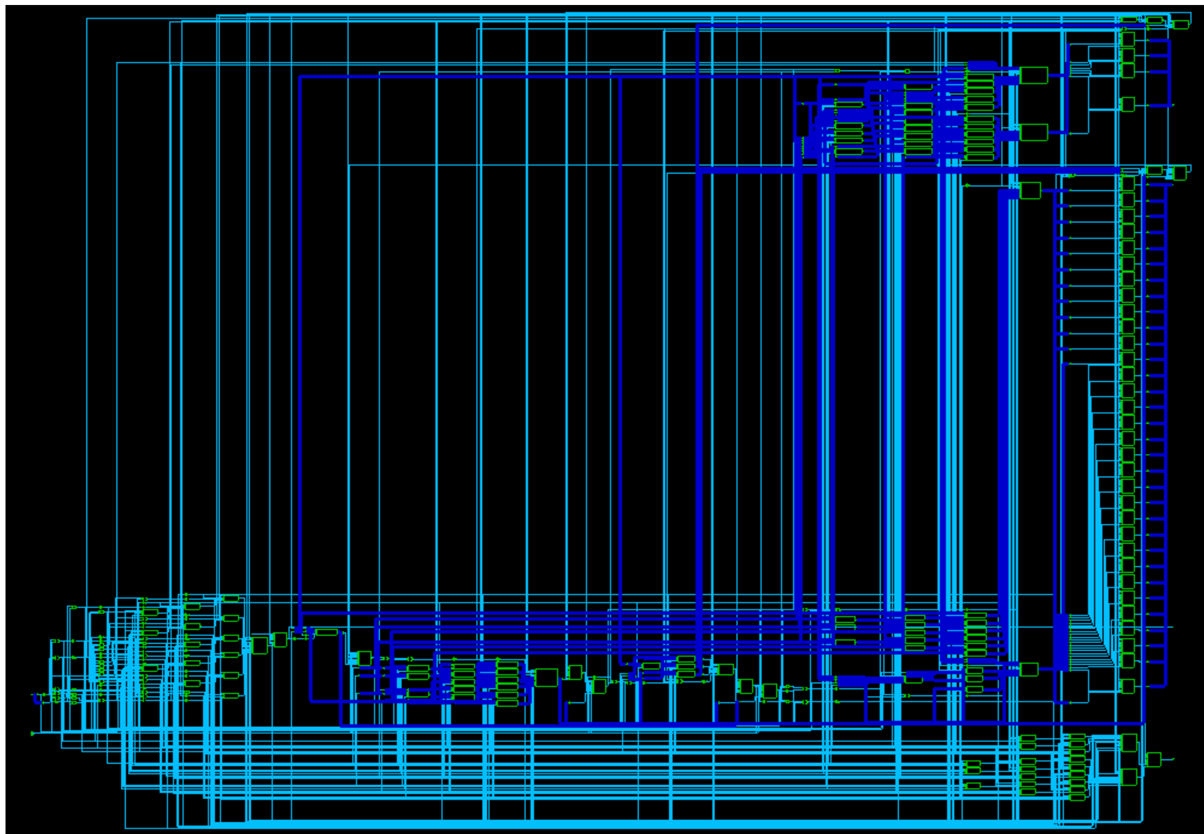


Figure 4.22: RTL of 2 bit parallel FSM based Huffman Decoder for ISCAS'89 S400 circuit

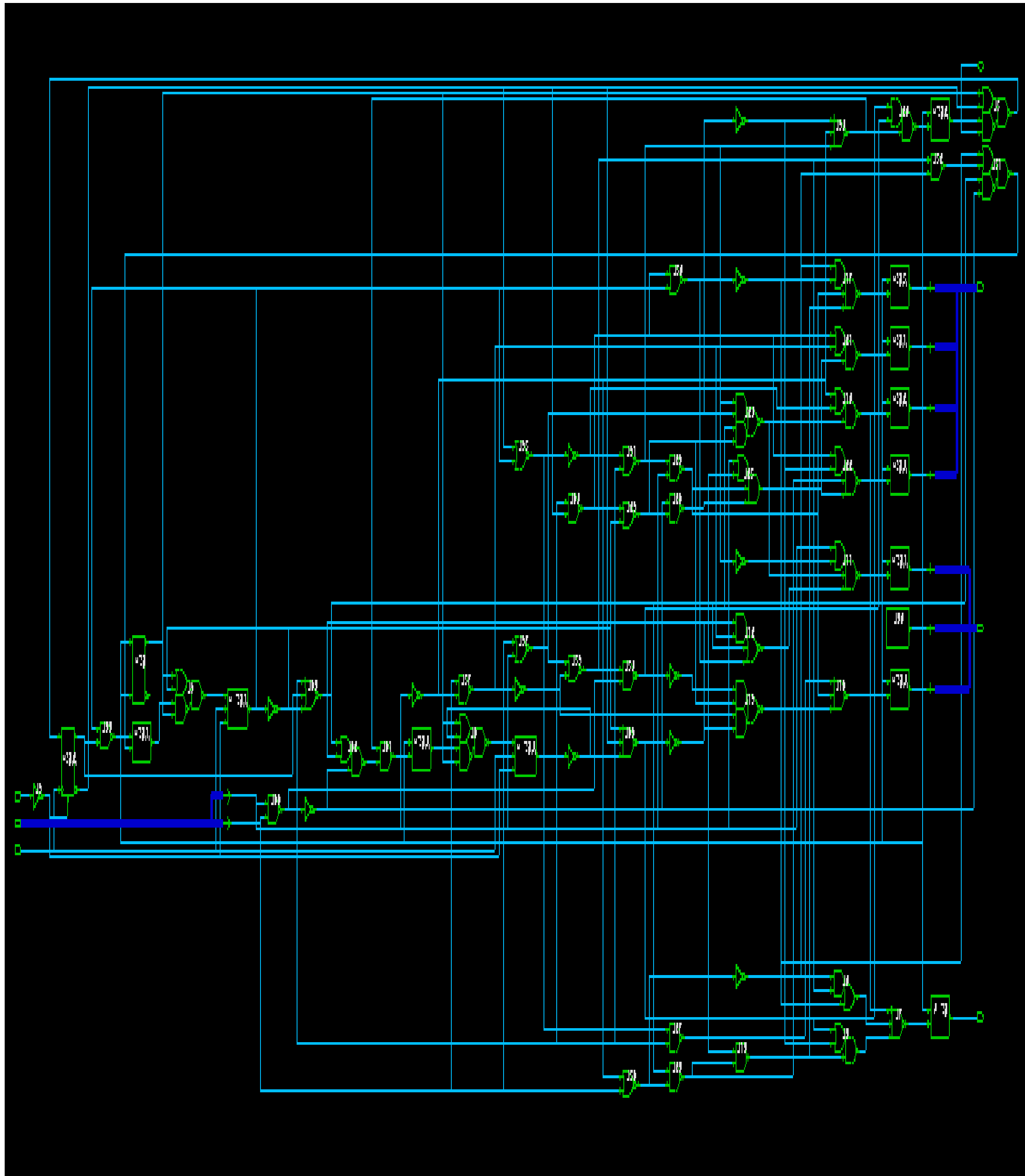


Figure 4.23: 2 bit parallel FSM based Huffman Decoder for ISCAS'89 S400 circuit after synthesis

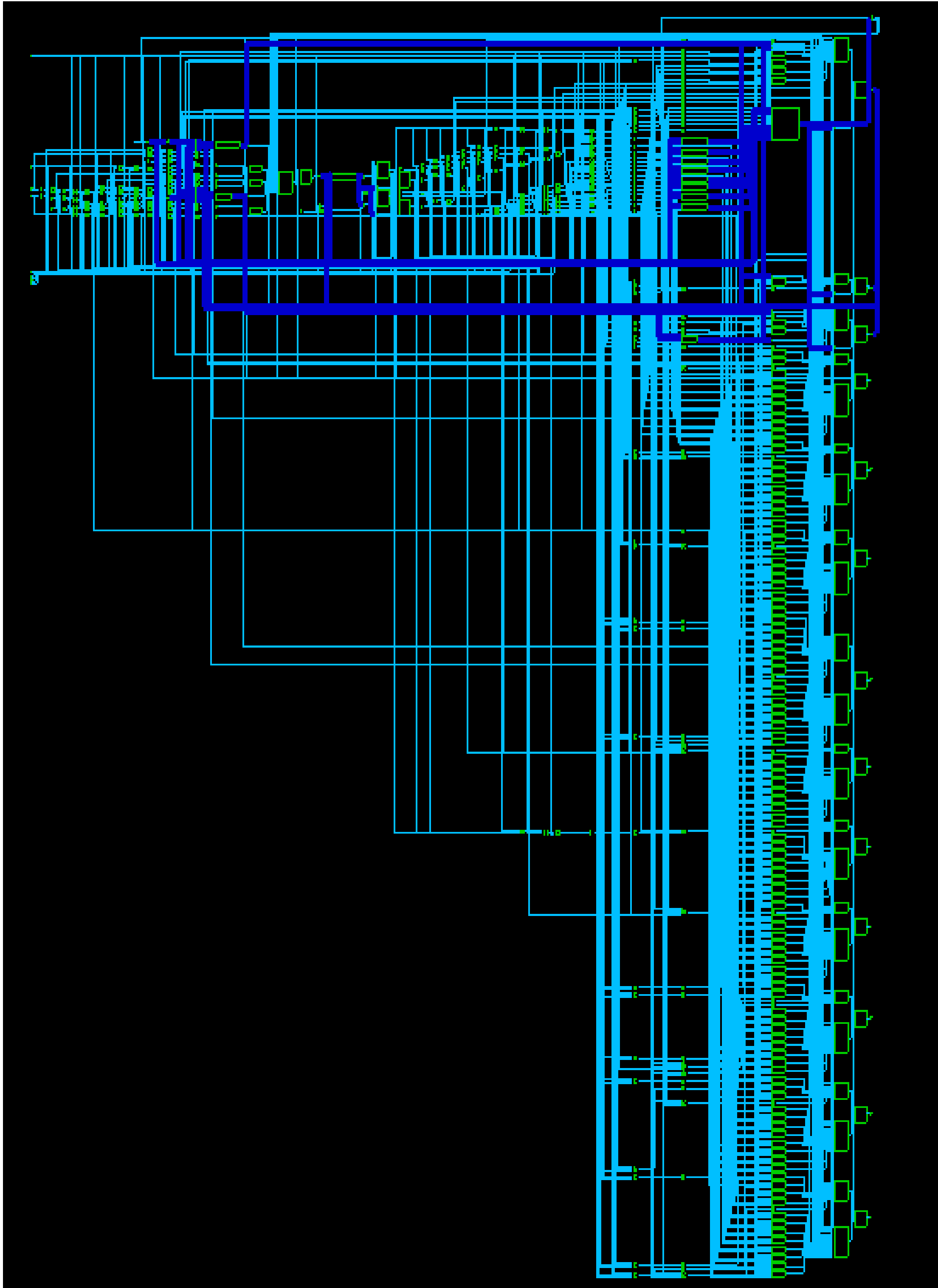


Figure 4.24: RTL of MCC Decoder

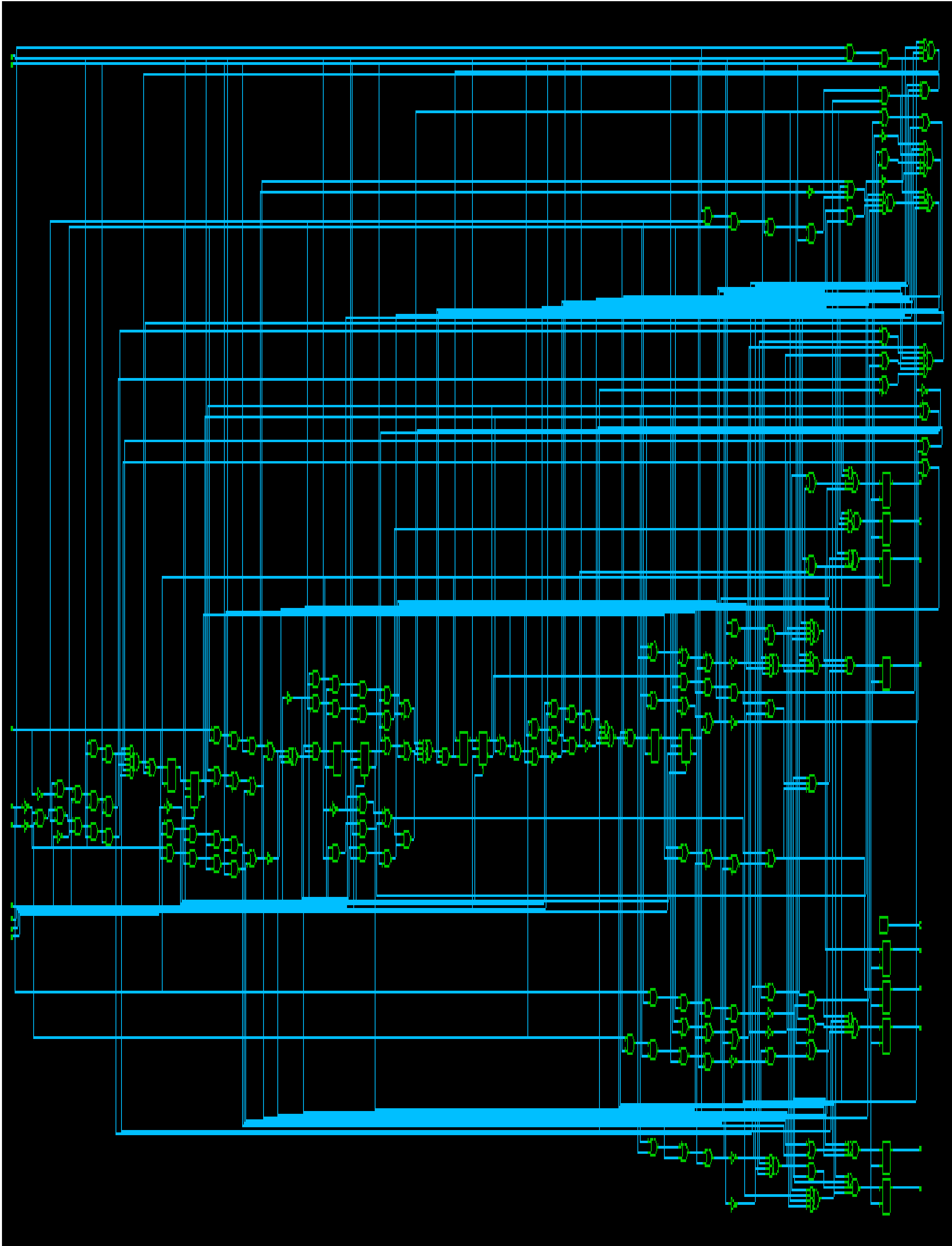


Figure 4.25: MCC Decoder after Synthesis

Figure 4.26: Block Diagram of FSM based MCC Decoder

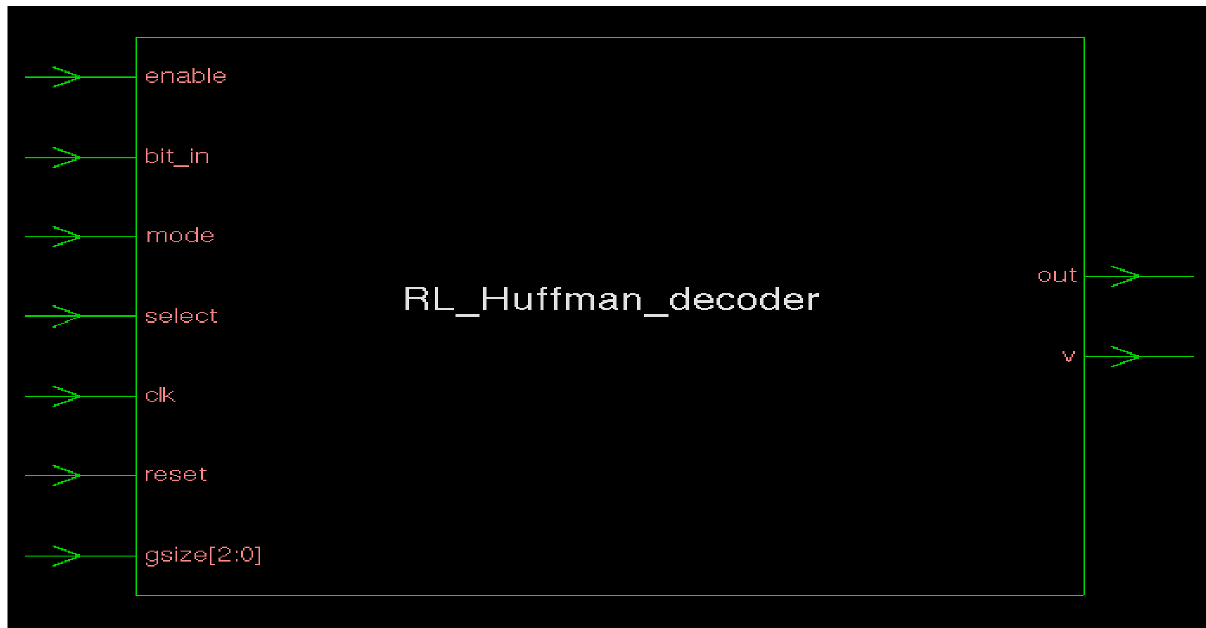


Figure 4.27: Block Diagram of Run Length plus Huffman Decoder

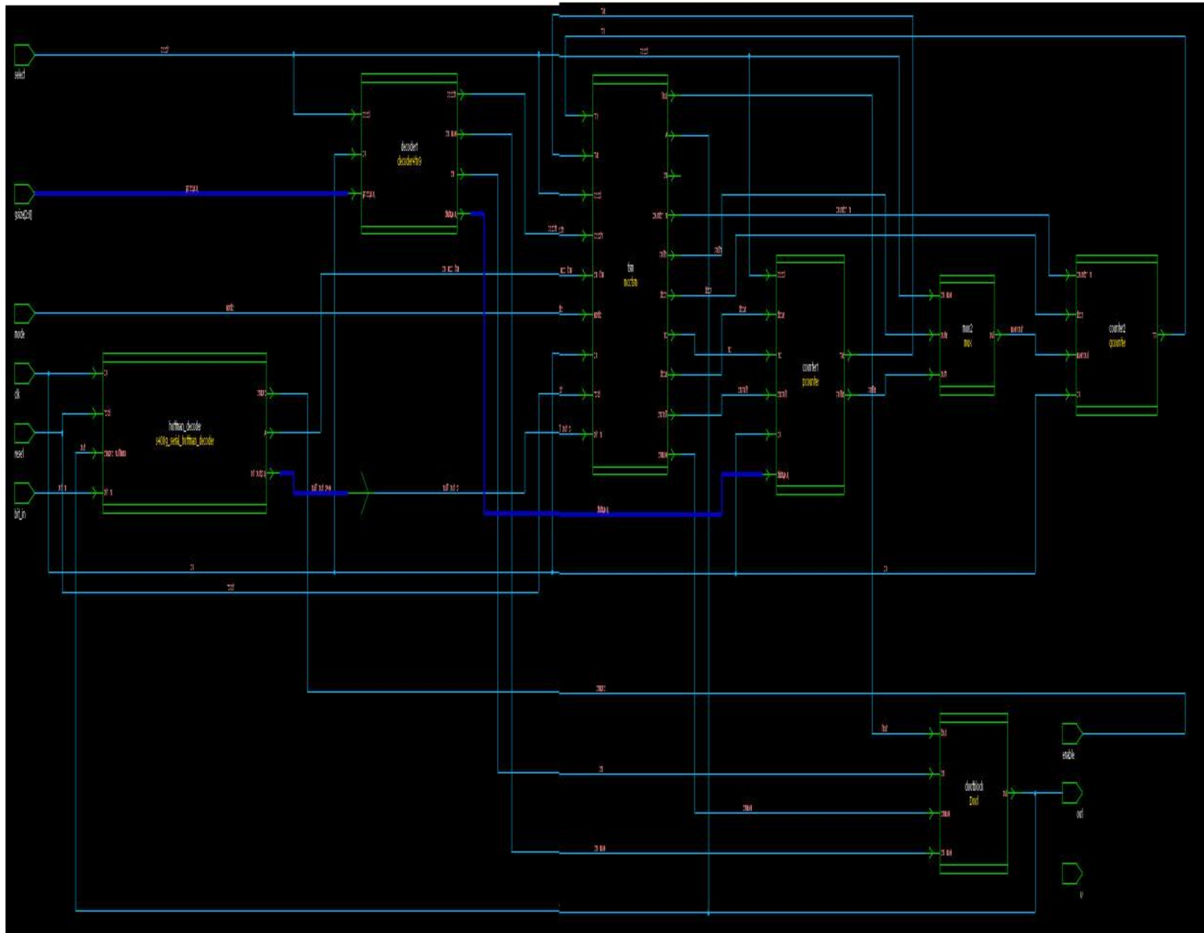


Figure 4.28: RTL of Huffman plus Runlength Decoder

Combinational area = 4362.222168 um^2

Noncombinational area = 2421.720215 um^2

Total cell area: = 6783.940918 um^2

Dynamic Power Units = 1mW (derived from V,C,T units)

Leakage Power Units = 1pW

Cell Internal Power = 27.3902 uW (69%)

Net Switching Power = 12.1166 uW (31%)

Total Dynamic Power = 39.5068 uW (100%)

Cell Leakage Power = 24.2553 nW

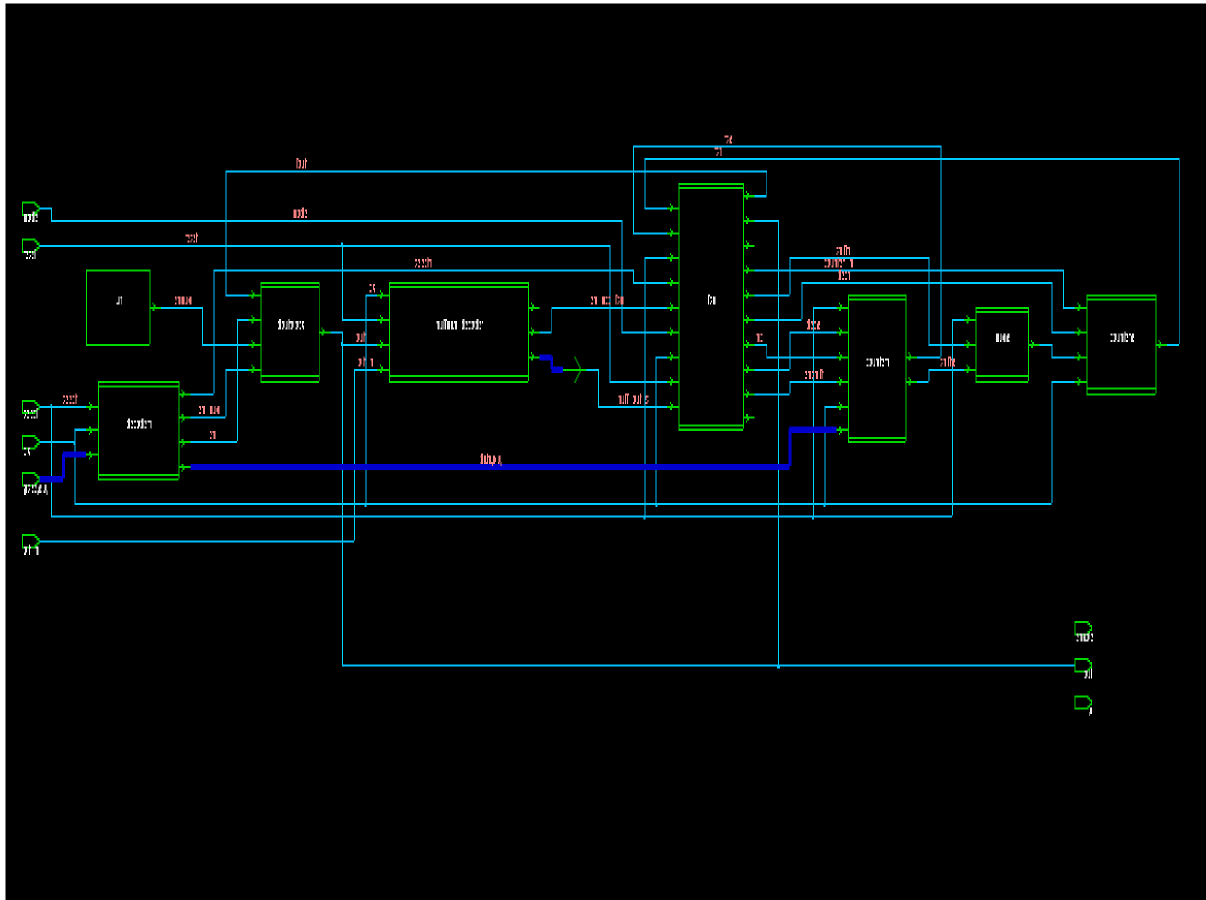


Figure 4.29: Run Length plus Huffman Decoder after Synthesis

4.3 Design Vision Results for Output Side Compaction

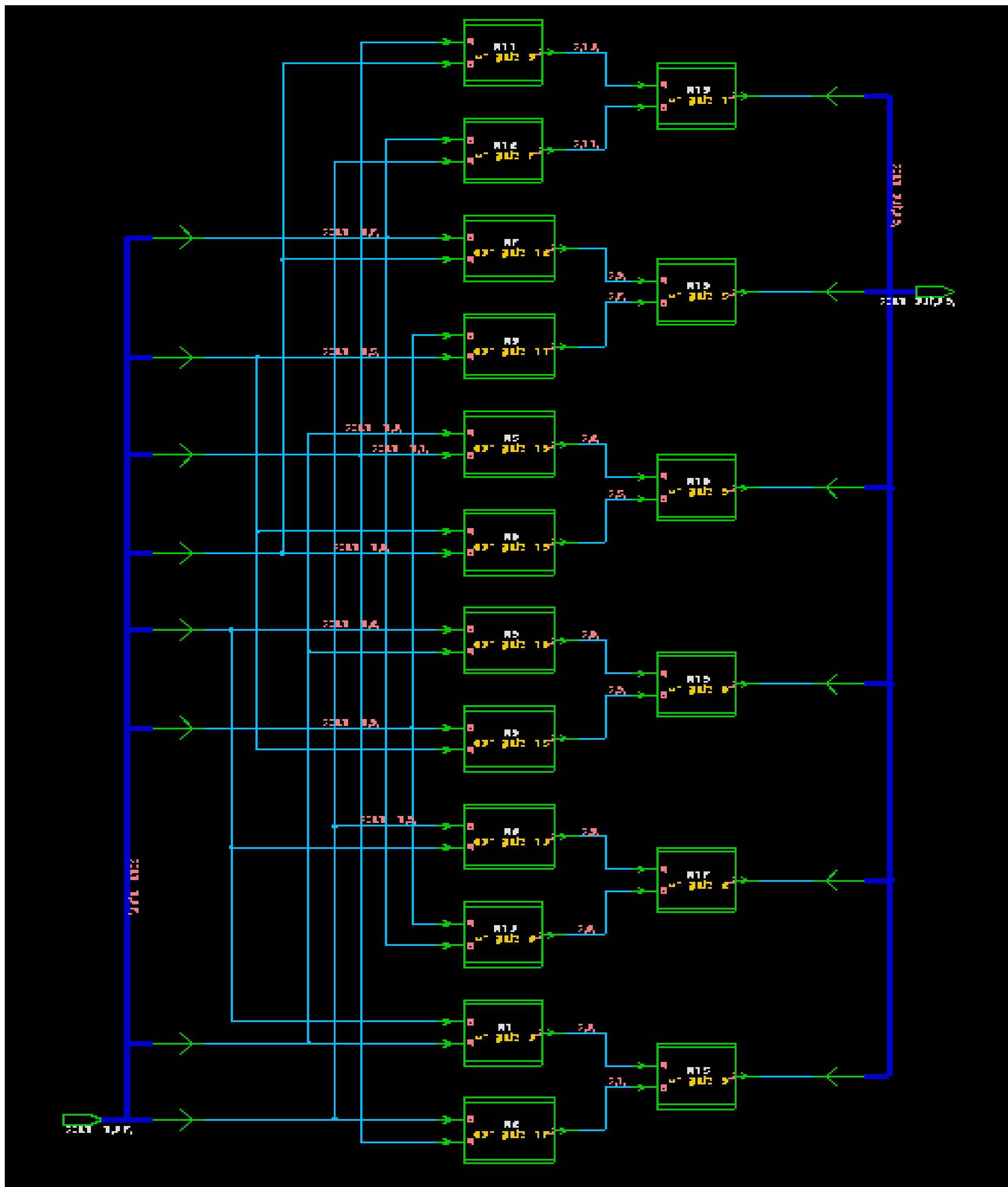


Figure 4.30: Synthesized X-compactor with 12 scan outs compacted to 6 scan outs

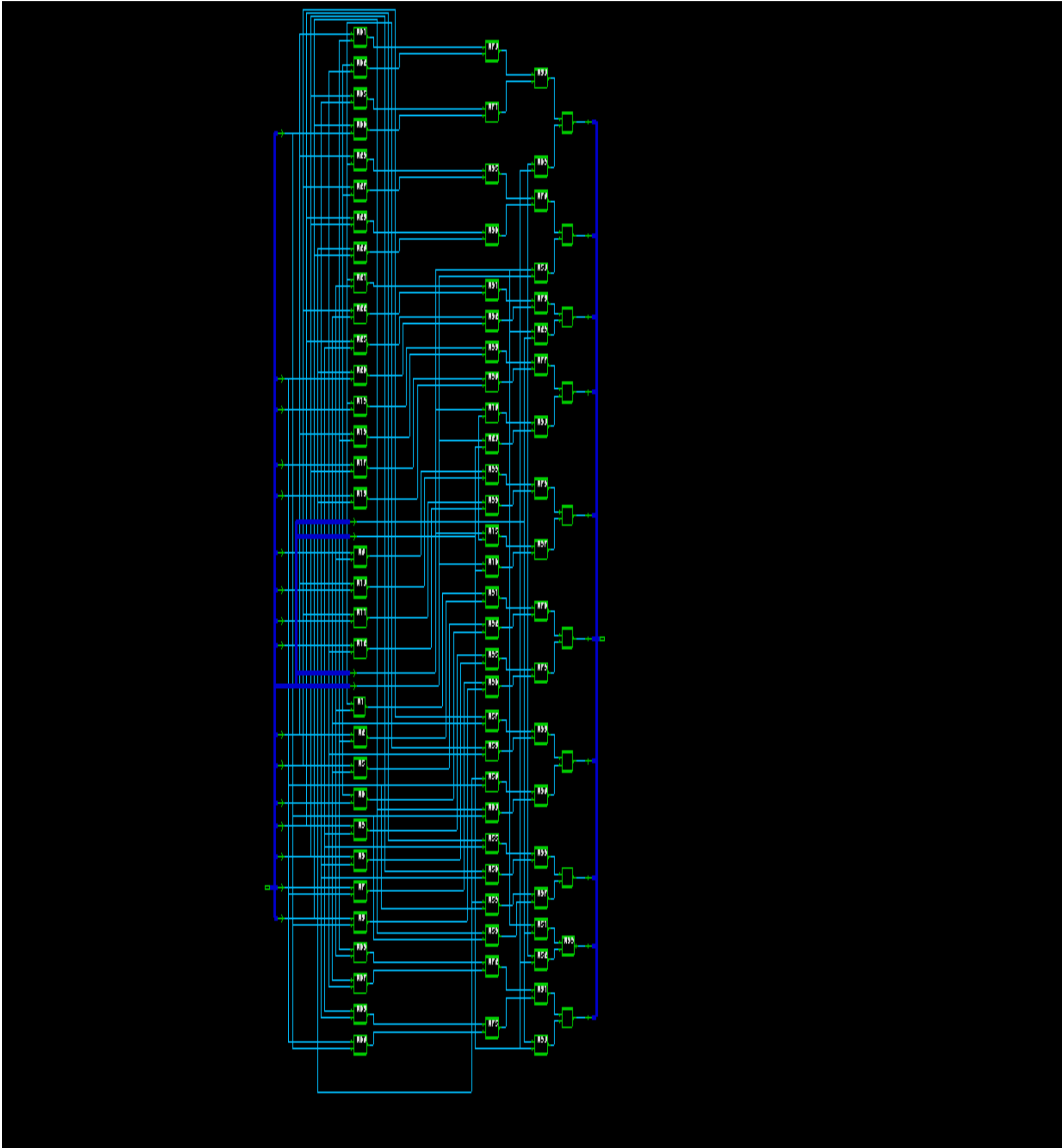


Figure 4.31: Synthesized X-compactor with 20 scan outs compacted to 10 scan outs

CHAPTER 5- CONCLUSION AND FUTURE WORK

A test compression method and decompression architecture for testing embedded cores in a SoC have been presented in thesis report. Run length based compression algorithms integrated with huffman compression scheme has been used with Double Hamming Distance Reordering scheme to have better compression efficiency and to save ATE memory and testing time.

The on chip decompressor is small and easy to implement. In addition it is scalable and independent of the core under test and utilizes only 41 Flip Flops with total cell area of 5262.163086 um^2.

Experimental results for ISCAS89 benchmarks show that the compression scheme along with bit stuffing and double hamming distance reordering technique and difference vector give better results than run length codes individually. Also decoders applied in the decompression engine have area over head that is not of much a problem.

The future work is to integrate heterogeneous codes to obtain higher compression ratio and to study the timing specifications of these circuits.

REFERENCES

[1] Takahiro Yamaguchi1, Marco Tilgner2, Masahiro Ishida1, and Dong Sam Ha3, "An Efficient method for compressing data," *International Test conference*, 1997.

[2] Eichelberger, E.B., T.W. Williams, "A Logic Design Structure for LSI Testability," *Proc.of Design Automation Conference*, pp. 462-468, 1977.

[3] Volkerink, E., A. Khoche, K. Hilliges, "Test economics for multi-site test with modern cost reduction techniques," *proc. of VLSI Test Symposium*, pp. 411–416, 2002.

[4] Eichelberger, E.B., E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," *IBM J. Res. Develop.,* pp. 262-272, 1983

[5] Zuying Luo et. al., "Test Power Optimization Techniques for CMOS Circuits," *Proceedings of the 11th Asian Test Symposium,* 2002.

[6] K. Royand, S. Prasad, "Low Power CMOS VLSI Circuit Design," *Wiely Inc*, 2000.

[7] Mitra S., K. S. Kim, "X-compact: An Efficient Response Compaction for Test Cost Reduction," *Proc. of International Test Conference*, pp. 311-320, 2002.

[8] Bardell, P.H., W. H. McAnney, J. Savir, "*Built-in Test For VLSI: Pseudo-random Techniques,*" Wiley Inter-Science, USA, 1987.

[9] Abhijit Jas, J. Gosh-Dastidar, M. Ng, and Nur A. Touba., "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, Vol. 22(6), pp. 797–806, June 2003.

[10] V. Iyengar, Krishnendu Chakrabarty, and Brian T. Murray, "Built-In Self Testing of Sequential Circuits Using Precomputed Test Sets," *VLSI Test Symposium*, pp. 418–423, 1998.

[11] Abhijit Jas, J. Ghosh-Dastidar, and Nur A. Touba "Scan Vector Compression/Decompression Using Statistical Coding,", *VLSI Test Symposium,* pp.114–120, 1999.

[12] Mehrdad Nourani and Mohammad Tehranipour, " RL-Huffman Encoding for Test Compression and Power Reduction in Scan Application", *ACM Trans. Design Automat. Electron. Syst.*, Vol. 10(1), pp. 91–115, January 2005.

[13] Anshuman Chandra and Krishnendu Chakrabarty, "System-on-a-Chip Data Compression and Decompression Architecture Based on Golomb Codes". *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.,* Vol. 20(3), pp. 355–368, March 2001.

[14] Anshuman Chandra and Krishnendu Chakrabarty, " Test Data Compression and Test Resource Partitioning for System-on-a-Chip Using Frequency-directed Run-length (FDR) Codes," *IEEE Trans. Comput.,* Vol. 52(8), pp. 1076–1088, August 2003.

[15] Anshuman Chandra and Krishnendu Chakrabarty, " A Unified Approach to Reduce SOC Test Data Volume, Scan Power, and Testing Time," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.,* 22(3): pp. 352–363, March 2003.

[16] M. Burrows and D. Wheeler, "Block Sorting Lossless Data Compression Algorithm," *System Research Center, Research Report 124, Digital System Research Center*, Palo Alto, CA, May 1994.

[17] M.R. Nelson, "Data Compression with the Burrows Wheeler Transformation," *Dr. Dobb's Journal*, pp. 46-50, September 1996.

[18] Usha S. Mehta, Kankar S. Dasgupta, Niranjan M. Devashrayee, "Hamming Distance Based Reordering and Columnwise Bit Stuffing with Difference Vector," *in 23rd International Conference on VLSI Design*, 2010.

[19] Francis G. Wolff and Chris Papachristou, "Multiscan-Based Test Compression and Hardware Decompression Using LZ77," *ITC'02: International Test Conference,* pp. 331–339, 2002.

[20] K. Loudon, Mastering Algorithms with C. O' Reilly, 1999

[21] A.-H. El-Maleh, and R.-H. Al-Abaji, "Extended Frequency-Directed Run-Length Code With Improved Application to System-on-A-Chip Test Data Compression," *in 9th International Conference, Electronics, Circuits and Systems*, vol. 2, pp. 449 – 452, Sept. 2002.

[22] Ya-Nan Wen, Guang-Huei Lin and Sao-Jie Chen, "Optimal Mutiple-Bit Huffman Decoding," *in IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20,pp. 621-623, May 2010.